AFRL-RI-RS-TR-2017-241

A PLATFORM FOR CONTEXTUAL MOBILE PRIVACY

INTERNATIONAL COMPUTER SCIENCE INSTITUTE

DECEMBER 2017

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the 88th ABW, Wright-Patterson AFB Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2017-241 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
FRANCES A. ROSE
Work Unit Manager

/ S /
JOHN D. MATYJAS
Technical Advisor, Computer
& Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
DEC 2017	FINAL TECHNICAL REPORT	JUN 2016 – JUL 2017

4. TITLE AND SUBTITLE

A PLATFORM FOR CONTEXTUAL MOBILE PRIVACY

5a. CONTRACT NUMBER
N/A

5b. GRANT NUMBER
FA8750-16-C-0140

5c. PROGRAM ELEMENT NUMBER
DHS

6. AUTHOR(S)

Serge Egelman, Nathan Good

5d. PROJECT NUMBER
DHSB

5e. TASK NUMBER
IC

5f. WORK UNIT NUMBER
SI

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
International Computer Science Institute (ICSI)
1947 Center Street, Suite 600
Berkeley, CA 94127

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RITE
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2017-241

12. DISTRIBUTION AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited. PA# 88ABW-2017-5993
Date Cleared: 28 Nov 2017

13. SUPPLEMENTARY NOTES

14. ABSTRACT

We developed a system that balances the privacy needs of users and organizations when using personal devices in the workplace-- "Bring Your Own Device" (BYOD) environments. In so doing, we performed qualitative interviews with extreme users, to under- stand their privacy needs, the shortcomings of current systems, and their existing coping mechanisms. Based on these interviews, we developed a system that applies machine-learning to automatically infer when access to sensitive data is likely to be expected by the user. We performed a field study to collect real-world training data to train the classifier offline. In parallel, we performed an online study to evaluate designs for a user interface (i.e., a "privacy management dashboard"). Based on our study results, we implemented our designs into the Android platform and performed a subsequent field study to validate our designs.

15. SUBJECT TERMS

Privacy, smartphones, BYOD, usability

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**FRANCES A. ROSE**
U	U	U	UU	90	19b. TELEPHONE NUMBER (*Include area code*) **N/A**

TABLE OF CONTENTS

Section **Page**

List of Figures

List of Tables

Acknowledgements

This work is a collective effort involving members of ICSI's Usable Security and Privacy Group, the Berkeley Laboratory for Usable and Experimental Security (BLUES) and Good Research: PI Serge Egelman, Irwin Reyes, Joel Reardon, Primal Wijesekera, Jennifer Chen, and Nathan Good. We would like to thank Program Manager Anil John, along with Ryan Triplett and Frances Rose for their guidance and support.

1 SUMMARY

In this project, we sought to understand how users and organizations balance privacy needs when using personal devices for both work and personal purposes— that is, "Bring Your Own Device" (BYOD) environments. Our goal was to better understand the challenges users and organizations faced so that we could develop a better privacy management system that addresses their needs. Specifically, we performed qualitative interviews among "extreme users" (those in environments who have particularly complicated needs) to answer the following questions:

- What do users want control over with respect to privacy in an organization?
- How should these controls be implemented?
- How can we build systems to make automated decisions in order to reduce user burden and facilitate better choices in-line with users' expectations?

In answering these questions, we interviewed 15 employees working in law enforcement organizations (i.e., the California Department of Justice and Los Angeles District Attorney's offices). Interviewees were in a range of roles with very particular privacy needs (e.g., prosecutors, investigators, and administrators). We found that users had sophisticated privacy needs that were not being met, particularly surrounding the access to personal information, location data, photos/media, and contact information, when using their smartphones in the workplace.

In addressing these users' needs, we developed a modified version of the Android platform that allows users to control how information is collected by third-party applications. The heart of our system is a trained classifier, which makes real-time decisions about whether an application should have access to certain protected data. Additionally, we provided specific functionality to address the areas that users found most problematic:

- For contacts, we provided a separate public/private primitive, which allows users to prevent applications from accessing professional or personal contacts, when appropriate.
- When denying access to certain data types, our system instead provides "fake" or less granular data, which allows privacy to be preserved, while limiting negative impacts on application functionality.
- For access to photos/media, we developed filters to identify and optionally remove potentially-sensitive features, before that media is shared with applications, in accordance with the end users' wishes.

We performed an initial field study using a sample of 131 Android users. We applied

the Experience Sampling Method (ESM) [11], wherein we periodically prompted participants about third-party applications' recent accesses to protected data. The prompts asked participants whether they would have allowed or denied the access, if given the choice. Alongside these prompts, we collected contextual data about what applications had accessed data, the types of data, and other situational information. We used the responses to these prompts to train a classifier using the contextual data as a feature set. We showed that this classifier was able to make much better decisions about access to sensitive data, reducing the error rate over the existing system by four-fold.

In parallel to this work, we developed a user interface. The theory behind the user interface is that if machine-learning is being used to automatically infer users' privacy preferences and making decisions based on those inferences, an interface is needed to show users what decisions have been previously made. This serves two purposes: transparency and control. In addition to showing the users what decisions have been made, it also allows them to alter those decisions, thereby retraining the classifier (and further reducing its error rate). We evaluated this design using an online study, which showed it to be usable.

Finally, we implemented both the classifier and the user interface into a working version of the Android platform and performed another field study to validate our designs. We gave instrumented phones running our system to 37 participants who used it for a one-week period. We corroborated our previous findings, in that error rates were reduced by 50-75% over the privacy management system.

In addition to our working system, the result of this work were three academic publications [18, 19, 17], and an additional publication in submission. These publications and the publication in submission address the research questions above, covering findings from our interviews as well as innovations in the user experience, and the technology to manage privacy requests at the OS-level, as well as improvements and integration of our classifier.

These results demonstrate that when provided the technology we have developed, end-users can effectively manage complex privacy decisions and trust the system to handle their private information in-line with their expectations. The implications of this work are that BYOD solutions that are end-user controlled can potentially increase adoption of these systems while simultaneously addressing joint user-organizational concerns around privacy and information security.

2 INTRODUCTION

Mobile phones are increasingly used as general purpose computers, with users taking advantage of a great number of apps to help with communication, productivity, organization, and diversion. Frequently these apps are nominally free, but the developers make a profit through the collection of sensitive user data. For example, Google Play's current top two flashlight apps (i.e., apps that turn on and off a phone's

camera flash) require access to the user's location, and previously requested address book contacts. This access to private data is not needed to actually operate the app itself, but instead is only used for monetization purposes.

These resources are highly sensitive. In interviews we conducted, we found that users, particularly those in law enforcement professions find the idea of delivering your entire address book to a third-party entity to be unacceptable. Such users are not alone. In fact, both Android and iOS provide a permission system as a means to protect users' private data. Arbitrary apps cannot simply access all user data they must first request access and the user then grants it. This is the application of a fundamental principle in security: the principle of least privilege. This means that apps should only be given the power and capabilities that they need to work and nothing more. This mitigates the damage caused by malicious behavior, both accidental and intentional.

Originally, Android permissions were presented as install-time ultimatums that were unsuccessful at achieving their goals [9, 6]. More recently, Android started using an *ask-on-first-use* (AOFU) model. In AOFU, the user is explicitly asked— at runtime— whether to grant or deny a permission via a dialog box, the first time an app attempts to access the protected resource. This approach gives the user a little more contextual information: e.g., it may be curious that a text messaging app needs to use the microphone, but knowing that the request occurred after the user tried to use a speech-to-text feature clarifies the likely rationale.

By design, however, AOFU takes the user's decision at one moment and then uses it in perpetuity for all future requests from that app for that permission, unless the user navigates several layers of settings to change it. During the course of this project, we showed that this method is highly error prone [19]: it mis-predicts the users' preferences resulting in privacy violations. This is because AOFU is not a correct model for user behavior and decision-making [18, 19]. AOFU fails to account for the contexts in which future requests may arise. Users are nuanced and they vary their decisions based on a variety of factors, such as the visibility of the requesting application (i.e., whether it was actively in use when it requested a permission), what the user was actually doing at the time, as well as a variety of other factors.

As part of this project, we implemented and evaluated the usability of a novel mobile privacy management system that builds heavily on our prior theoretical work, as well as the interviews with "extreme users"—those with very serious privacy concerns when using their mobile devices in professional environments— that we conducted to better understand the problems that users face. To resolve the longstanding challenges of mobile privacy management, we applied machine-learning (ML) to dynamically manage app permissions, and then we proposed a user interface design to help users manage that system [17]. This work applies Nissenbaum's theory of Privacy as Contextual Integrity [14]. We then implemented these systems on the Android platform and performed a field study to evaluate their effectiveness at aligning app privacy

behaviors with users' expectations. The machine-learning (ML) model runs entirely on the device and uses infrequent user prompts to retrain and improve its accuracy over time. When the ML model makes a mistake, the user interface is available to support the user in reviewing and modifying privacy decisions, thereby retraining the ML.

We performed a 37-person field study to validate our new privacy management system, measuring its efficacy and how it interacted with participants and third-party apps. We issued each participant a smartphone running a custom Android OS with our permission system that used an online classifier, which participants used as their primary phones for a one-week study period. This produced real-world usage data from 253 unique apps, which corresponded to more than 1,100 permission decisions. Overall, participants denied 24% of permission requests. Our data show that AOFU matched participant privacy preferences only 80% of the time, while the new contextual model matched preferences 95% of the time, reducing the error rate by 75%.

3 METHODS, ASSUMPTIONS AND PROCEDURES

Our approach was iterative, engaging end-users through user experience research methodologies (using observational studies, interviews, and surveys) and evolving our solution through system development (building user interfaces, the modified android operating system, and a classifier). Each of the iterative cycles of our project was divided into stages, with results from each stage feeding into both the research directions and technical goals for the next stage. Stages of our iterative research process, along with the methods employed at each stage are described below.

Stage 1 Understand user needs and requirements gathering

Interviews of Stakeholders and *Extreme Users* Develop Initial Requirements from Privacy Concerns Development and test Lo-Fi prototypes

Stage 2 Prototyping and Testing

Develop and survey of Hi-Fi Prototypes of user control dashboard
Implementation privacy controls
Interviews of Stakeholders and *Extreme Users* Refinement of Classifier

Stage 3 Implementation and Field Testing

Implementation and refinement of the classifier
Implementation and refinement of the Android OS
Deployment and field test of devices Interviews of field test users

Below we describe the details of the methods we employed.

3.1 User Interviews

We conducted a series of interviews of 15 users who had mobile devices and experiences with either BYOD currently or in the past had both a personal and work phone. For the initial interviews we focused on *Extreme Users*; users who represented more sophistication and sensitivity to privacy concerns than the average consumer. Focusing on Extreme Users is a methodology that is employed in User Experience research, particularly when exploring issues such as privacy that are not typically on most users mind. The Stanford Design School (D.SchooL) describes research on Extreme Users as follows [15]:

Designers engage with users (people!) to understand their needs and gain insights about their lives. They also draw inspiration from their work-arounds and frameworks. When you speak with and observe extreme users, their needs are amplified and their work-arounds are often more notable. This helps you pull out meaningful needs that may not pop when engaging with the middle of the bell curve. However, the needs that are uncovered through extreme users are often also needs of a wider population.

Using Extreme Users would allow us to uncover issues and concerns that would likely overlap with end-user concerns of the larger population, but would be buried or difficult to uncover in the course of interviewing laymen. For our methodology, we used Extreme Users to uncover the issues that were most important; we used that knowledge, along with our own experience and previous work, to motivate our system implementation; and finally we went back to the larger population to verify and confirm our insights.

The *Extreme Users* that we recruited were from various government agencies (California Department of Justice, Los Angeles District Attorney, etc.) and in roles as either prosecutors (civil and criminal), investigators or law enforcement where the separation of private and personal information was very important to their jobs and in some cases personal safety and safety of the individuals that they worked with was of great importance.

Each interview we conducted was either over the phone or in-person, and lasted between 1-1.5 hours total. The initial interviews were semi-structured, with a script that the moderator walked through with the users, with time for unscripted questions at the end. These initial interviews focused on their current practices and concerns, and explored ways in which these concerns could be addressed by technology.

The next iteration of interviews also focused on concerns, but for the later portion of the interview they were shown a working prototype that incorporated some of the earlier suggestions (e.g., location and address book controls), as well their understanding and opinions on our user interface (the privacy "dashboard").

The last round of interviews we performed on subjects who were not Extreme Users, but from a larger population from our field study. We interviewed users initially about their phone use, and again at the end of our field study about their experiences with our devices and using the controls.

3.2 Initial Classifier Design

Our goal was to collect a feature set that would allow the platform to automatically detect the context surrounding each permission request, so that once trained, the platform can use these features to infer whether a permission request is likely to be deemed appropriate by the user. This entails determining the factors that define a context, as well as the factors that are indicative of users' privacy preferences. Defining context is important because it scopes privacy decisions: whenever the context in which an application requests access to a particular data type changes, the system needs to make a decision about whether or not to grant access. Similarly, in order to make these decisions automatically without prompting the user, this entails determining the factors that predict users' privacy preferences.

Our main objective was to explore the learnability of user privacy preferences over prompting for every sensitive permission request; an effectively-trained permission system should be able make decisions on behalf of the users accurately with minimal user involvement. While prompting for every sensitive permission request gives the user a finer degree of control over their privacy protection, we showed that it is impractical due to the sheer volume of requests that would result [18]; we believed the best way forward was to learn a users' privacy preferences. Learning involves three essential components:

- The platform needs to figure out which permission requests are likely to defy user expectations.
- The platform also needs to figure out which permission types are more sensitive on a per-user basis.
- The platform needs to learn what other observable factors are used in users' decision processes and how they can be used to infer users' decision-making.

We used the Experience Sampling Method (ESM) to collect ground truth data about users' privacy preferences [11]. ESM involves repeatedly questioning participants *in situ* about a recently observed event; in this case, we probabilistically asked them about an application's recent access to data on their phone, and whether they would have permitted it if given the choice. We treated participants' responses to these ESM probes as our main dependent variable (**Figure 1**). We instrumented the Android platform so that these prompts would periodically appear—no more than once per day—after recent permissions requests (**Table 1**).

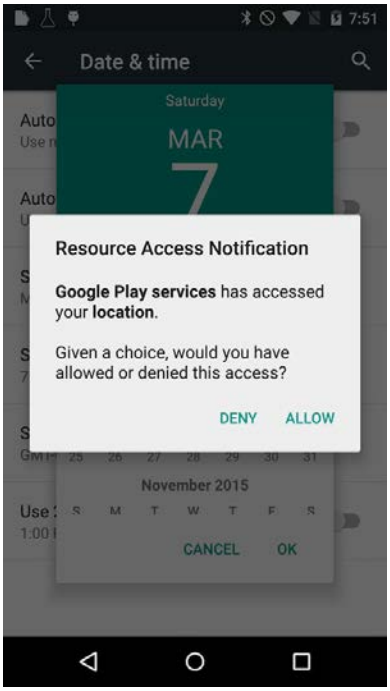


Figure 1: A screenshot of an ESM prompt.

Table 1: Participants in the first study received prompts when applications attempted to use these permissions

Permission Type	Activity
ACCESS WIFI STATE	View nearby SSIDs
NFC	Communicate via NFC
READ HISTORY BOOKMARKS	Read users' browser history
ACCESS FINE LOCATION	Read GPS location
ACCESS COARSE LOCATION	Read network-inferred location (i.e., cell tower and/or WiFi)
LOCATION HARDWARE	Directly access GPS data
READ CALL LOG	Read call history
ADD VOICEMAIL	Read call history
READ SMS	Read sent/received/draft SMS
SEND SMS	Send SMS
*INTERNET	Access Internet when roaming
*WRITE SYNC SETTINGS	Change application sync settings when roaming

We also instrumented participants' smartphones to obtain data about their privacy-related behaviors and the frequency with which applications accessed protected resources. The instrumentation required a set of modifications to the Android operating

system and flashing a custom Android version onto participants' devices.

Table 2 contains the complete list of behavioral and runtime events our instrumentation recorded. The behavioral data fell under several categories, all chosen based on several hypotheses that we had about the types of behaviors that might correlate with privacy preferences: web-browsing habits, screen locking behavior, third-party application usage behavior, audio preferences, call habits, camera usage patterns, and behavior related to security settings. For example, we hypothesized that someone who manually locks their device screen are more privacy-conscious than someone who lets it time out.

Table 2: Instrumented events that form our feature set

Type	Event Recorded
Behavioral Instrumentation	Changing developer options
	Opening/Closing security settings
	Changing security settings
	Enabling/Disabling NFC
	Changing location mode
	Opening/Closing location settings
	Changing screen-lock type
	Use of two factor authentication
	Log initial settings information
	User locks the screen
	Screen times out
	App locks the screen
	Audio mode changed
	Enabling/Disabling speakerphone
	Connecting/Disconnecting headphones
	Muting the phone
	Taking an audio call
	Taking a picture (front- vs. rear-facing)
	Visiting an HTTPS link in Chrome
	Responding to a notification
	Unlocking the phone
Runtime Information	An application changing the visibility
	Platform switches to a new activity
Permission Requests	An app requests a sensitive permission
	ESM prompt for a selected permission

The primary purpose of recording user behaviors was to observe how much time a user voluntarily spends on making security and privacy related decisions by changing default settings or re-visiting decisions they have made earlier (security settings, location settings), because observing these behaviors could be indicative of their privacy

preferences. For instance, the choices they had made about screen locks; how careful they are with their web browsing habits, such as how often do they use Chrome incognito tabs, how often they get warnings from visiting suspicious websites, and so forth. We also collected other observable traits that could be indicative of privacy preferences, such as how often they take pictures, their audio preferences, and how active they are with audio calls using the phone.

Finally, we collected runtime information about the context of each permission request, including the visibility of the requesting application at the time of request (i.e., whether it was in the foreground or background), what the user was doing when the request was made (i.e., the name of the foreground application), and the exact Android API function invoked by the application to determine what information was requested. The visibility of an application reflects the extent to which the user was likely aware that the application was running; if the application was in the foreground, the user had cues that the application was running, but if it was in the background, then the user was likely not aware that the application was running and therefore might find the permission request unexpected—some background services can still be visible to the user due to on-screen notification or other cues that could be perceptible. We monitored processes' memory priority levels to determine the visibility of all Android processes. We also collected information about which Android `Activity` was active in the application, which indicates the UI elements exposed to the user.

Further details on the results and methodology are described in the Appendix and in [19].

3.3 Prototype and User Testing

We tested various stages of the user interface and the notification interaction with the classifier, as part of our methodology. We performed remote user testing of the classifier with a walk-through, talk aloud protocol on a small set of 10 initial users, and then performed a large scale survey and test of the user interface on 400, and subsequently 580 users. A detailed description of our test process and results are included in the Appendix and in the paper [17].

3.4 System Implementation

We implemented a complete ML pipeline that includes: mechanisms to allow users to review and redress their decisions based on the results of our UI proto typing study [17]; ways to mask resource denial from apps so that apps continue to run, even when permissions are denied (unless those permissions were critical to their functionality); and finally, a classifier that takes surrounding contextual signals to predict user preferences for each permission request. This means our usability study is a more accurate assessment of how the system behaves in the wild than the previous investigations, which relied on user expectations rather than consequential privacy decisions. This final study validates that prior work.

3.4.1 A Local SVM Classifier

We previously implemented an offline classifier at the onset of this project and suggested this could be deployed as a remote web-accessible service in order to shift compute costs from the mobile device to a more powerful dedicated server [19]. We note, however, that this design required sending privacy-relevant data beyond the smartphone, which creates a larger attack surface and increases system costs and complexity. It also creates significant security risks if the server responsible for making decisions is compromised or is trained with spurious data.

To mitigate these security and privacy issues, we implemented and integrated the full SVM classifier into the Android operating system as a system service. We ported the open-source implementation of *libsvm* to Android 6.0.1 (Marshmallow) [4], and built two additional system-level services to interface with the SVM: the *SVMTrainManager*, which trains the model using user-provided privacy preferences through prompts (See **Figure 2**); and the *PermissionService*, which uses the SVM to regulate applications accessing permission-protected resources and issues a prompt for the user for cases when the model produces low-confidence predictions. The *SVMTrainManager* notifies the *PermissionService* when the model is trained and ready for use. These two new services are implemented into the core Android operating system, and neither are accessible to third-party apps. On average, model training takes less than 5 seconds. We instrumented all Android control flows responsible for sharing sensitive permission-protected data types to pass through this new pipeline.

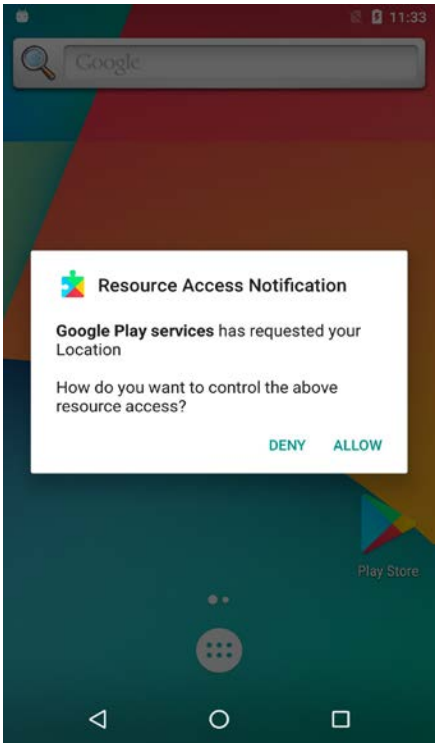


Figure 2: A screenshot of a permission request prompt.

3.4.2 Bootstrapping

We deployed our trainable permission system along with a generic model that was pre-trained with the real-world permission decisions of 131 users, shared with us from our initial work [19]. This ensured that a new user has an initial model for making privacy decisions. This initial model, however, is inadequate for accurately predicting any particular individual user's preferences, because it simply has no knowledge of that particular user. Despite that, we previously showed that our model only needs 12 additional user-provided permission decisions before the model attains peak accuracy. Given this, our implemented system requires that the user make 12 decisions early on to train the initial model to that particular user's preferences.

The initial 12 decisions are selected based on weighted reservoir sampling. We weigh the combination of *application:permission:visibility*[1] by the frequency that these are observed; the most-frequent combinations are the likeliest to produce a permission request prompt (**Figure 1**). The intuition behind this strategy is to focus more on the frequently occurring permission requests over rarer ones. We used these same prompts for validating our classifier during the field study.

3.4.3 Feature Set

Our model considers the name of the application requesting the permission, the application in the foreground at the time of the request (if different than the application making the request), the requested permission type (e.g., Location, Camera, Contacts), and the visibility of the application making the request. In a pilot study, our system implemented the full feature set we previously used [19].

This design, however, resulted in a noticeable reduction in device responsiveness as reported by multiple study participants. We subsequently removed the "time of request" feature for the second phase of our study. The removal of the time feature from the ML enabled the platform to cache higher number of ML decisions saving reducing the overhead stemming from running the ML for each different permission request.

3.4.4 Sensitive Resources

Previous work by Felt et al. argued that certain permissions should be presented as runtime prompts, as those permissions guard sensitive resources whose use cases typically impart contextual cues indicating why an app would need that resource [8]. Beginning with Android 6.0 (Marshmallow), the OS designated certain permissions as "dangerous" [10], and prompts the user to grant or deny permission when an app tries to use it for the first time. The user's response to this prompt then carries forward to all future uses of that resource by the requesting application.

Our experimental permission system uses both Felt's set of recommended permissions for runtime prompts and Android's own "dangerous" ones. We did, however, opt to omit

[1] "application" is the app requesting the permission, "permission" is the requested resource type, and "visibility" denotes whether the user is made aware that the app is running on the device.

a few permissions from the resulting set that we viewed as irrelevant to most users. The INTERNET and WRITE SYNC SETTINGS permissions were discounted, as we did not expect any participant (all recruited locally) to roam internationally during the 1-week study period. We eliminated the NFC permission because previous work demonstrated that very few apps operate on NFC tags. Our system ignores the READ HISTORY BOOKMARKS permission, as this is no longer supported.

We extended our initial framework [18, 19] to monitor and regulate all attempts by apps to resources protected by any of the 24 permissions we monitored. We avoid false positives by monitoring both the requested permission and the returned data type.

3.4.5 Permission Denial

Making changes to the permission system carries the risk of app instability, as apps may not expect to have their resource requests denied [7]. If denying permissions results in frequent crashes, then users may become more permissive simply to improve app stability. We therefore designed our implementation with this concern in mind: rather than simply withholding sensitive information in the event of a denied permission, our system supplies apps with well-formed but otherwise non-private "spoofed" data. This enables apps to continue functioning unless access to the permission-protected resource is critical to the app's correct behavior.

For example, if an app requests access to the microphone, but our permission system denies it, the app will still receive a valid audio stream: not an actual signal from the microphone, but that of a pre-recorded generic sound. (In our implementation we used a loop of a whale song). This design allows apps to operate on valid data while still preserving user privacy.

Permission-protected databases (e.g., contact lists and calendars) require finer- grained regulation under our permission system. For instance, an app may have a legitimate need to access the contact list. Under the stock Android permission system, an app is either able to read *all contacts* or *no contacts*. We improve upon this by adding a notion of *provenance* to each entry: every contact list item contains a field that records the app that created the entry. If our permission system denies an app access to the contact list, the app is still able to write into the contacts database and read back any entries that it previously created. Apps without these database permissions are effectively placed in a sandbox, in which they can still carry out valid operations on their own versions of the data. They neither produce an exception nor obtain all the information in the database. We allow full access to the databases only to apps that are granted the appropriate permission.

3.4.6 Contextually Aware Permission Manager

We recognize that our classifier is bound to make mistakes. Therefore, it is crucial to provide a mechanism for users to review and amend decisions made by the permission model on their behalf. Mobile operating systems have configuration panels to manage

app permissions, but these fail to provide users key information or options to make informed decisions. However, our study that examined user interface prototypes [17] proposed a new interface to solve this problem [17]. In that study (still a part of this project), we evaluated several designs using interactive online mock-ups and found that the design significantly improved user experience over the stock configuration panel. We followed those recommendations in the design that we built as part of this implementation.

We built our contextual permission manager as a system-space app, similar to Android's *Settings* app (**Figure 3**). Our permission manager has three main objectives: (i) to display all recent permission requests and the corresponding "allow" or "deny" decisions from the ML model; (ii) to allow users to review and change app permissions; and (iii) to display all the resources an app can access.

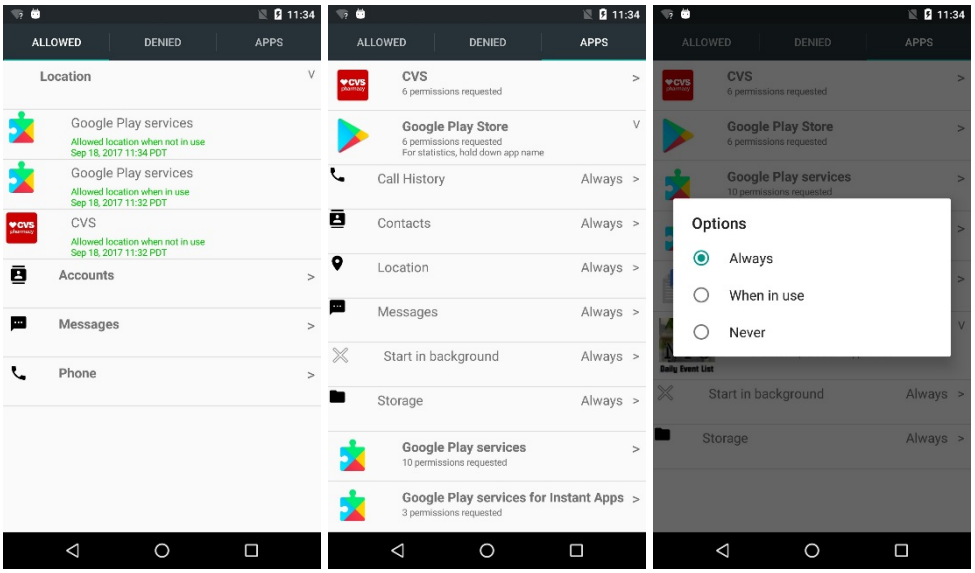


Figure 3: The recent-allowed app activity (left), a list of installed apps and their associated permissions (center). Permissions can be *always* granted, granted only *when in use*, or *never* granted (right).

When users set preferences(rules) in the permission manager, before making a ML decision, platform checks to see if the user has set any rules for the current request; if a match is found, rather than going to the ML, platform will use the current rule to respond to the permission request accordingly. The system does not use these user-set rules to train the ML model, it is hard to capture the contextuality behind these changes so the platform cannot create any of the contextual features to train the ML.

3.5 Validation Study Methodology

We tested our implementation by performing a field study with 37 participants. Our goals were to understand how third-party apps and end-users react to a more restrictive

and selective permission model, as compared to the default AOFU model.

For a period of one week, each participant used a smartphone (Nexus 5X) running a custom version of the Android OS (a variation of Android 6.0.1) built with the new permission system detailed in the previous section. During the study period, all of a participant's sensitive data was protected by the new contextually-aware permission model.

3.5.1 Participant's Privacy Preferences

We used the Experience Sampling Method (ESM) to understand how participants want to control certain sensitive resource accesses [11], similar to how we used it in our earlier work [19]. ESM involves repeatedly questioning participants *in situ* about a recently observed event; in our case, the event is an app requesting access to a sensitive resource. We probabilistically asked them about an application's recent request to access to data on their phone, and how they want to control future similar requests (Figure 1). We treated participants' responses to these ESM prompts as our main dependent variable, which we used to validate the accuracy of the decisions that the classifier was automatically making.

Each participant during the study period responded to 4 prompts per day, and at most one per hour. The prompting was divided into two phases. The first phase was the *bootstrapping phase*, which we described earlier, to train the classifier. The second phase was the *validation phase*, which was used to measure the accuracy of the ML model. In addition to the validation phase prompts, participants might also have occasional prompts for low-confidence decisions made by the ML; a detailed discussion on low-confidence decisions is provided later. During our study period, only 4 participants ever experienced low-confidence prompts.

3.5.2 Recruitment

We recruited participants in two phases: a pilot in May 2017 and the full study in August 2017. We placed a recruitment ad on Craigslist under "et cetera jobs" and "domestic gigs."[2] The title of the advertisement was "Smartphone Research Study," and it stated that the study was about how people interact with their smartphones. We made no mention of security or privacy. Interested participants downloaded a screening app from the Google Play store, which asked for demographic information and collected their smartphone make and model. We screened out applicants who were under 18 years of age or used CDMA providers, since our experimental phones were only GSM-compatible. We collected data on participants' installed apps, so that we could pre-install free apps prior to them visiting our laboratory. (We only encountered paid apps for a few participants, and those apps were installed once we setup their Google account on the test phone.)

We scheduled a time with participants who met the screening requirements to do the

[2] The study was approved by our Institutional Review Board.

initial setup. Overall, 63 people showed up to our laboratory, and of those, 61 qualified (2 were rejected because our screening application did not identify some CDMA carriers). The initial setup took roughly 30 minutes and involved transferring their SIM cards, helping them set up their Google and other accounts, and making sure they had all the applications they used. We compensated each participant with a $35 gift card for showing up.

During the pilot phase, out of 20 people who were given phones, 14 participants had technical issues with the phone preventing them from using it, leaving only 6 participants with usable data. During the main phase, out of 42 people who were given phones, we had the following issues:

- 4 participants misinterpreted our ESM prompts so we filtered out their prompt responses;
- 5 participants suffered from a bug in the code that inhibited the validation phase of the ML;
- 2 participants performed factory resets on the phone before returning it, which destroyed stored logs.

This left 31 participants with usable data from the main phase. We combined the 6 participants with usable data from the first phase with the 31 from the second phase to produce our sample of 37 users, since we did not alter the study between phases. All our results are drawn from log data and interview responses from those 37 users. Of that population, 21 were female and 16 were male; ages ranged from 18 to 59 years old ($\mu = 34.25, er = 9.92$).

After initial setup, participants used the experimental phones for one week in lieu of their normal phones. They were allowed to install, use, and uninstall any apps that they wanted. Our logging framework kept track of every protected resource accessed by an app, along with the contextual data surrounding each application request. All the logged events were stored compressed in the local system.

3.5.3 Exit Interview

When participants returned to our laboratory, we first copied the log data from the phones to make sure that they had actually used the phone during the study period. We then administered a semi-structured exit interview, which had four components:

- **New Permission Manager UI**—We asked participants to show us how they would use the UI (Figure 3) to block a given application from accessing background location data, as well as how difficult they found it. We also checked our data to see how they interacted with the UI during the study period, and asked them about the circumstances for those interactions. The objective of this task was to validate the design objectives of the UI, including whether they use it to resolve issues stemming from resource denial.

- **Permission Prompts**—We asked participants questions about permission prompts they had encountered during the study. We asked why they allowed or denied permission requests and also how they felt about the prompts. We asked them to rate their experience with the prompts across 3 different categories: levels of surprise, feelings of control, and to what extent they felt the new system had increased transparency. The objective of this section was to understand the impact of the runtime prompts.
- **Permission Models**—We asked participants questions about their perspectives on the privacy protections in Android. We asked how much they understood the current system. We then explained our new system, and asked how they felt about letting ML act on their behalf. The objective of this section was to understand how much participants actually understood the new permission model.
- **Privacy Concerns**—Finally, we asked participants how they usually make privacy decisions on their mobile devices, how serious they are about privacy, and how much are they willing to pay for privacy. We also asked demographic questions.

Three researchers independently coded 144 responses to the *Permission Prompts* and *Permission Model* questions (the other questions involved either direct observations or reporting participants' responses verbatim without the need for coding). Prior to meeting to achieve consensus, the three coders disagreed on 17 responses, which resulted in an inter-rater agreement of 86.43% and Fleiss' kappa yielded 0.747, indicating substantial agreement.

After the exit survey, we answered any remaining questions, and then assisted them in transferring their SIM cards back into their personal phones. Finally, we compensated each participant with a $100 gift card.

4 RESULTS AND DISCUSSION

At the end of the study period, we collected 1,159 privacy decisions (prompt responses) from 37 participants. A total of 133 unique applications caused prompts for 17 different sensitive permission types. During the study period, 24.23% of all runtime prompts were denied by participants. Most (66%) of these prompts occurred when the requesting application was running visibly. Our instrumentation logged 5.4M sensitive permission requests originating from 253 unique applications for 17 different permission types. On average, a sensitive permission request occurred once every 4 seconds.

In the remainder of the paper, we describe the shortcomings of the existing ask-on-first-use permission model, both in accuracy and in aligning with users' expectations; we show how our proposed system has vastly greater accuracy in inferring users' privacy preferences and applying them towards regulating application permissions; and we

show that is does this with minimal impact on app functionality. Finally, we present results from the exit interviews regarding participants' perceptions about the training prompts and the privacy management user interface.

4.1 Status Quo Problems

In the "ask-on-first-use" (AOFU) model, the user is prompted only the first time an app attempts to access a protected resource. Requesting these permissions at runtime allows the user to infer the potential reason for the request, based on what they were doing at the time (i.e., context). AOFU's shortcoming, however, is that it naïvely reapplies the user's first-use decision in subsequent scenarios, without adapting to different contexts. Our previous work showed that failing to account for changing contexts produces high error rates (i.e., the user would have opted to deny permission if AOFU had not granted it based on the first-use prompt) [18].

We note that our initial work on this project measured the accuracy of the AOFU model by merely collecting users' responses to runtime permission prompts, without actually enforcing them by denying apps access to data [19]. Thus, the accuracy rates reported by that study may not actually be valid, since users may elect to change their permission-granting preferences, if they result in a loss of application functionality. Thus, we evaluated the performance of the AOFU approach (in current use by Android and iOS) by presenting participants with permission prompts that *actually* resulted in the denial of application permissions.

During the study period, each participant responded to combinations of *application:permission* more than once. As AOFU is deterministic, we can simulate it by comparing a user's first response to an *application:permission* combination to future responses to the prompts for the same app and permission. We use this data to measure how often AOFU matches the user's preference in subsequent requests.

Our data show that the AOFU permission model has a median error rate[3] of 20%: in more than one-fifth of app requests for permission-protected resources, participants changed their initial response for the same *application:permission* combination. Of the 37 participants, 64% had at least one such discrepancy between the first-use and subsequent preferences. This refutes AOFU's core assumption that only few users will deviate from their initial preferences in future cases. This observation corroborates the initial study [19], in which 79% of 131 participants were shown to deviate from their initial responses in subsequent cases.

The errors shown in AOFU, could be either privacy violations or losses of functionality. A privacy violation occurs when the system grants an app access to a protected resource, contrary to the user's preference, had she been prompted.

Loss of functionality occurs when the permission system denies access to a protected

[3] We report medians because the error rate was not normally distributed among participants.

resource, which the user would have otherwise permitted. We consider privacy violations to be the more severe type of error, as the user is unable to take back sensitive information once an app has acquired it and transmitted it to a remote server. However, loss of functionality is still undesirable because those errors might incentivize the user to be overly permissive in order to regain that functionality. From our data, we found that 66.67% of AOFU errors were privacy violations; the remaining 33.33% were losses in functionality.

4.1.1 AOFU User Expectations

Errors in permission systems could arise from a variety of reasons. Mismatched user expectations and lack of comprehension are two critical ones, which could hamper any permission model's utility. User comprehension is critical because users may make suboptimal decisions when they do not fully understand permission prompts, hindering the ability of the permission system to protect sensitive system resources. Users must be able to comprehend the decision they are making and the consequences of their choices. Recent work on AOFU has examined the motives behind users' decisions and how it varies between categories of applications, as well as how people adapt their behavior to the new model [3, 2, 1].

In our study, the participants had, on average, 5 years of experience with Android. This indicates that most of our participants have experienced both install-time permissions—the permission model prior to Android 6.0, released in 2015—and runtime "ask-on-first-use" permission prompts. The majority of participants said they noticed the shift to AOFU prompts, and they were aware that these prompts are a way to ask the user for consent to share data with an app. A large minority of participants (40%), however, had an inadequate understanding of how AOFU works, which could substantially hinder that permission model's effectiveness in protecting user data.

Four out of the 37 participants expressed doubts about the rationale behind the prompts. Rather than seeing permission prompts as a way for users to regulate access to their sensitive data, these participants viewed these prompts as a mechanism to extract more information from them:

"When I see prompts, I feel like they want to know something about me, not that they want to protect anything." (P21)

One *possible* explanation is that some users grew accustomed to install-time prompts, and subsequently perceived the change to runtime prompts as a new way for Android to collect user data. Although it is impractical to project how prevalent this sentiment is in the general population, we cannot reject its existence. Hence, more work is needed to measure its impact and explore the potential solutions.

A third (31.4%) of our participants were not aware that responding to an AOFU prompt results in a blanket approval (or denial) that carries forward to all the app's future uses of the requested resource. Most participants believed that responses were only valid for

a certain amount of time, such as just for that session or just that single request. This misconception significantly hinders AOFU's ability to correctly anticipate the user's preferences in future occurrences. Again, this observation raises the question of whether users would respond differently if they had a more accurate understanding of how AOFU works:

"[I] didn't know that granting a permission carries forward in the future until otherwise changed. [I] expected permissions to be for just that one use." (P25)

It is clear that granting blanket approval to sensitive resources is not what users expect all the time. On the other hand, had our participants been asked for their input on every permission request, they would have received a prompt once every 4 seconds—involving the user more frequently has practical limitations. How, then, can we best project users' privacy preferences to future scenarios without overwhelming them with prompts?

4.2 Classifier Accuracy

During the week-long study period, each participant was subject to two operational phases of the contextual permission system: (a) the initial *learning phase*, where participant responses to prompts were used to re-train the SVM classifier according to each individual's preferences, and (b) the steady-state *validation phase*, where responses to prompts were collected to measure the accuracy of the classifier's decisions.

As previously discussed in our section on bootstrapping, we use weighted reservoir sampling during the learning phase to prioritize prompting for the most commonly observed instances of *application:permission:visibility* combinations. During the validation phase, participants received the same prompts for random combinations of features. This ensured that we collected validation results both for previously-encountered and new combinations. We placed a maximum limit of 3 prompts per combination in order to further improve prompt diversity and coverage. After presenting participants with prompts, the instrumentation recorded the response and the corresponding decision produced by the classifier. Using participant responses to prompts as ground-truth, we measured the classifier's accuracy during the validation phase. From our sample of 37 participants, we had to exclude 6 of them due to a cache coherency bug that was discovered after the pilot, which degraded classifier performance. For the remainder of this section, our results are drawn from the remaining sample of 31, unless otherwise noted.

Taken as a whole, these 31 participants responded to 640 total prompts in the validation phase. Our contextual permission model produced a median accuracy of 90%, compared to 80% under AOFU for the same population. The classifier reduced AOFU's error rate by 50%, with the majority of classifier errors consisting of privacy violations (i.e., access granted when the user would have denied it).

4.2.1 Offline Learning

We were curious whether the accuracy of our system could be improved through the use of offline learning, which would require much more computing power. Using participant responses to permission prompts, we analyzed how an offline SVM classifier would perform. We implemented the SVM model using the *KSVM* module in *R*. We performed this analysis on data from all 37 participants, using leave-one-out cross-validation to evaluate how the offline classifier would perform for each participant.

The offline model had a median accuracy of 94.74% across the 37 participants. By comparison, AOFU had an 80% accuracy for the same population. This represents a 75% error reduction in the offline contextual model compared to AOFU. These numbers corroborate our prior findings [19]. We stress the significance of this corroboration, because the results hold in the presence of actual resource denial, which was not examined in the prior study. This suggests that users will continue to indicate their true preferences in response to prompts, even when those preferences are enforced, potentially resulting in unanticipated app behavior.

We note the accuracy difference between the SVM classifier we integrated into Android and the *R* model (90% vs. 94.74%, respectively). This is due to how the Android SVM implementation performs the bootstrapping. This issue is not inherent to integrating an SVM classifier into Android. An updated implementation has the potential to reach the maximum accuracy observed in the offline model.

4.2.2 Decision Confidence

In our initial investigation of classifier-based permission models, we proposed using decision confidence to determine for which *application: permission:visibility* combinations users should be prompted in the validation phase [19]. The rate of decision confidence is also a measure of the extent to which the classifier has learned the user's preferences. The authors suggested that if this rate does not decrease over time, then AOFU will likely be a better system for those users.

In addition to the prediction, our classifier also produced a class probability, which we used as the measure of decision confidence. The classifier produced a binary result (i.e., allow or deny) with a cutoff point of 0.5. A decision probability close to the cutoff point is a less confident result than one far from it. We used the 95% confidence interval as a threshold to determine which decisions were low-confidence and which ones were not.

Only 4 of our field study participants experienced low-confidence classifier decisions that caused a prompt to appear after the bootstrapping period. Each of these participants had just one such low-confidence prompt appear. These prompts retrained the classifier, so the lack of any subsequent low-confidence prompts indicates that the classifier produced high-confidence predictions for the same *application:permission:visibility* combination in future cases.

The lack of additional training prompts also suggests that users are less likely to become habituated to prompting. The 4 participants who each received one additional prompt saw a total of 13 prompts (including the 12 prompts during the training phase). The remaining 27 participants saw just the 12 training phase prompts. Had our participants been subject to AOFU instead of our contextual permission system, they would have received a median of 15 prompts each, with a quarter of the participants receiving more than 17. Instead, we achieved a 75% error reduction (80% vs. 94.74%) and reduced user involvement by 20% (12 prompts vs. 15) through the use of classifier-driven permissions, compared to AOFU.

4.3 Impact on App Functionality

Previous research has shown that many applications do not properly handle cases where they are denied permission to access a protected resource [7]. One core objective of our work was to measure how apps responded to a stricter permission model than AOFU. For example, the system will be unusable if it causes erratic application behavior, through the use of dynamically granted permissions.

In the field study, our platform instrumentation recorded each application crash and its corresponding exception message. This information allowed us to identify the *possible* root cause of the crash and whether it was related to resource denial.

We observed 18 different exceptions classes, such as SecurityException, RuntimeException, and NullPointerException. For the remainder of this section, we focus on SecurityExceptions, which is directly related to resource denials. Almost all (98.96%) of the recorded SecurityExceptions were observed on the devices of just two participants. Each of the remaining participants encountered, on average, 18 SecurityExceptions during the study period (i.e., roughly 3 SecurityExceptions per day per participant).

Almost all (99.93%) SecurityExceptions were from apps attempting to use the READ PHONE STATE permission, which is used to obtain the phone number. In the event of a READ PHONE STATE denial, we designed our implementation to not supply the app with any phone number data. We had considered supplying a randomly-generated phone number, but decided against it due to potential risks, if the generated number were a valid phone number belonging to someone else.

For other denials, we opted to supply apps with generated data to ensure their continued operation, without actually exposing private user data. During the study period, the classifier denied 10.34% of all permission requests; more than 2,000 denials per participant per day. Our implementation, however, only recorded an average of 3 SecurityExceptions per day per participant. This indicates that passing synthetic but well-formed data to apps in lieu of actual private user data does satisfy app functionality expectations to a great extent.

Our results are a positive sign for future permission systems more restrictive than the

current AOFU model: permissions can be more restrictive without forcing the user to trade off usability for improved privacy protection, as we will show in the next section. If apps gracefully handle resource denials, then users are free to specify their privacy preferences without risking functionality issues.

4.4 User Reactions to Prompts

The use of runtime prompts was initially proposed as a mechanism to obtain better-informed consent from users. At the end of the study period, we conducted exit interviews with each participant in order to determine the extent to which these assumptions were met.

We measured how much participants were surprised to see the prompts during the course of the study period (on a scale of 1="not surprised" to 5="very surprised"). Participants expressed an average rating of 2.7. Almost half (44%) of the participants indicated that the prompts surprised them, and among them, 70% were surprised at the frequency with which the prompts appeared (up to 4 times per day), though few participants expressed annoyance by that frequency (8.33%).

We asked participants to rate how much they felt that they were in control of resource usage (on a scale of 1="nothing changed compared to default Android" to 5="very much in control"). On average, our participants rated their experience as
3.44. Almost half (44%) of participants felt that they were in control of the system as a result of the prompts. A small number (14%) still felt helpless, regardless of their responses to the prompts. They felt resigned that applications would always obtain their data.

Finally, we asked participants how they felt about the transparency provided by the new system compared to their previous Android experiences (on a scale of 1="nothing changed" to 5="improved system transparency"). On average, participants rated system transparency in the middle (3). Almost half (47%) of them felt that the new system was more transparent. A minority (14%) mentioned wanting to know *why* apps were requesting particular sensitive data types.

From these observations, we believe that the new contextual permission system is a positive step toward improving user awareness. We believe this enables users to make better privacy decisions for themselves. Although additional work is needed to address some negative sentiments about the current implementation, this system has shown to be in the right direction overall.

4.5 User Reactions to Controls

Whenever an automated system makes decisions on a user's behalf, there is the inevitable risk that the system will make an incorrect decision. In our case this can cause apps to be over-privileged and risk privacy violations, or be under-privileged and risk app failure or reduced functionality. It is important to empower users so they can easily audit the decisions that were made on their behalf and to amend those decisions

that are not aligned with their preferences.

In our implementation, we built a user interface based on our prior validated prototypes [17]. This system allowed our participants to view automated permissions decisions made by the classifier, as well as set privacy preferences with respect to context (i.e., the visibility of the requesting app). We included this user interface as part of the operating system, as a panel within the system settings app.

When we on-boarded our participants, we mentioned to them that there was a new "permission manager" available, but to avoid priming them, we made sure not to emphasize it in any particular way. Our instrumented platform logged every time participants interacted with our permission manager to understand how they used it.

Fifteen of the 37 participants (40.5%) opened the permission manager during the study period. Our implementation logged a total of 169 preference changes across these participants. Only 6 out of 37 participants (16.2%) changed the settings to be *more restrictive*. Of the adjustments made towards more restrictiveness, the majority were for the GET ACCOUNTS permission, which prevents apps from reading the user's stored credential data (e.g., usernames linked to accounts on the device, such as for Google, Twitter, etc.). In contrast, the most-common permission that participants adjusted to be more permissive was READ CONTACTS. When asked for their motives behind these changes, the majority of participants said that functionality was their main reason for granting more access, and the sensitivity of data for restricting access.

We also asked participants to demonstrate how they would change the settings of a familiar app to only be able to access their location when they are using that app. We based this task off of one of the evaluation tasks we performed in the dashboard evaluation experiment [17], where we performed an online study to evaluate a low-fidelity prototype of the design on which we based our user interface. All but two of our participants were able to correctly complete this task using the user interface. Participants rated the average ease of the task as 1.15 (on a scale from 1="very easy" to 5="very hard"). We conclude that participants are able to understand the permission interface after having used it for a week, and without special instructions.

The permission manager also enables users to diagnose application crashes that result from a resource denial (a feature not present in the original design on which we based it). In exit interviews, we examined how participants responded to app crashes in their experiences with the device. The majority of participants reported that their first step was to restart the app that had crashed. If that was unsuccessful, they would then restart their phone. This informs the design of a future system: if an app crashes as a result of a resource denial, the platform should clearly communicate this to users or otherwise automatically adjust the permissions on their behalf. This could be communicated through a dialog or in the notification bar.

4.6 Discussion

The core objective of our 37-person field study was to analyze how a contextually-aware, more-restrictive permission model performs in the wild, thereby validating the work we had performed in this project. We examined how participants balanced their privacy preferences with app functionality. This measures the real-world applicability of predicting user privacy decisions with the help of contextual cues surrounding each permission request.

4.6.1 Consequential Denial

Overall, participants denied 24% of all prompted permission requests. This is a 60% reduction in denials compared to the results we attained at the onset of the project, when evaluating an offline classifier [19], which did not enforce the user's decision to deny a permission and prompted the user using only hypothetical language: "given the choice, would you have denied...?" The decreased denial rate we observed is therefore unsurprising given that participants were now actually making a tradeoff between functionality and privacy, instead of expressing the degree to which privacy is important to them. Our results show that even in the presence of consequential resource denial, contextual cues helped to predict users' privacy decisions and better aligned permission settings with their expectations, as compared to the status quo.

4.6.2 Ask on First Use

Our results corroborate our initial work [18, 19] in showing that AOFU's inability to capture the context surrounding users' decisions is a cause of AOFU's significant error rate, and based on our qualitative interviews, provides serious concerns when used in high-risk BYOD environments. We also found that a significant portion of participants do not have an adequate understanding of how AOFU works, which further limits AOFU's utility: 11 participants did not realize that their prompt responses for AOFU are taken as permanent decisions; and 4 participants interpreted the prompts as yet another mechanism for collecting user data instead of as a privacy-protection mechanism. While the actual impact of these inaccurate beliefs is yet to be explored, we believe that these issues need to be fixed in the future, in order to increase Android's ability to predict and protect user data effectively.

4.6.3 Implementation Limitations

While our new permission model reduces the number of mis-predictions compared to AOFU by 50%, our offline analysis shows that it has the potential to reduce mis-predictions by 75%. A further examination revealed that the performance difference is due to the bootstrapping of the training dataset in the implementation. We note that difference is not inherent to running a classifier in Android, and so simply modifying our implementation to use these improvements will allow it to achieve the same performance.

4.6.4 Purpose

While our new permission model outperforms AOFU, it still does not explain to the user *why* an app needs to use a permission. In our exit interviews, we observed that 14% of participants expressed the desire to know *why* apps made a request in the first place. Previous work has shown that app functionality is a key factor in permission decisions [3]. If users were properly informed of the functionality requirement behind a permission request, then they might be better positioned to make decisions that meet their privacy and functionality expectations.

We believe that there are ways to extend contextual permission systems by incorporating the actual purpose of the request. For example, after introducing AOFU permissions, Android started encouraging app developers to provide the reason behind their permission requests so that the user can include that in the decision making process [5]. Tan et al. [16] showed that similar prompts on iOS actually resulted in users being more permissive about granting permissions to apps. Similarly, prior work has attempted to use static analysis to automatically incorporate inferred purpose [13, 12].

4.6.5 Resource Denial

When deploying more-restrictive permission systems, it is important that apps continue to run without entering into an error state that results from a resource denial. Users should be able to select their privacy preferences with minimal disruption to their experience; apps must not be able to force an ultimatum by simply not functioning if a permission is denied. Indeed, some participants simply allow most permission requests because that ensures their apps run properly.

The platform, therefore, is responsible to ensure that apps handle resource denials gracefully. To their credit, when Android introduced AOFU, it implemented some permission denials to appear like a lack of available data or the non-existence of hardware, instead of throwing a `SecurityException`. In our implementation, we take the extra step of supplying apps with generic but well-formed data in the event of a denial. We observed that our participants tended to deny more permissions as they progressed through the study period (on average 20% denial in the learning phase versus a 26% denial rate during the validation phase). Those participants also experienced a low rate of app failures due to resource denials. In the future, platforms should implement measures to reduce functionality losses stemming from having stricter privacy preferences. Failing to do so might otherwise compel users to compromise on their privacy preferences for the sake of functionality.

4.6.6 Remedying Unexpected Behavior

Regardless of any mitigations to avoid app crashes, it is practical to assume that apps will crash when they fail to receive expected data under certain circumstances. One way to remedy this is to give users tools to adjust the behavior of the permission system,

such as being able to be more permissive to certain applications in certain contexts. This approach, however, assumes that (i) users accurately attribute a crash event to a resource denial, which may not always be the case, and (ii) users are sufficiently technical to identify which resource denial caused the crash. In our implementation of a new permission manager, we address the latter assumption by providing users a timeline of recent decisions made by the new permission system, which can be used to deduce the cause of a crash.

Our exit interviews showed that few participants would think to check the permission manager following an application crash, so clearly more work is needed here. With proposals for more-accurate and more-restrictive permission models, it is necessary to have usable mechanisms to deal with inevitable crashes due to resource denials. The platform should provide mechanisms either to help the user diagnose and resolve such crashes, or to automatically fix permissions on a temporary basis and give the user an option to make the fix permanent.

5 CONCLUSION

Our validation study shows how applications and users respond to a real-world deployment of a novel contextually-aware permission model, which we developed based on: i) soliciting feedback from "extreme users" in BYOD environment, ii) iterative prototyping on the user interface dashboard, and iii) the development of a privacy preferences classifier using offline training. The new permission system based on these components significantly reduced the error rate from that of the prevailing "ask-on-first-use" model first deployed in Android 6.0. While prior work already demonstrated ways to increase the protection provided by new permission models, we believe our study provides opportunities to further improve performance and address practical limitations in actual implementations.

6 REFERENCES

[1] P. Andriotis, S. Li, T. Spyridopoulos, and G. Stringhini. *A Comparative Study of Android Users' Privacy Preferences Under the Runtime Permission Model*, pages 604–622. Springer International Publishing, Cham, 2017.

[2] P. Andriotis, M. A. Sasse, and G. Stringhini. Permissions snapshots: Assessing users' adaptation to the android runtime permission model. In *2016 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6, Dec 2016.

[3] B. Bonné, S. T. Peddinti, I. Bilogrevic, and N. Taft. Exploring decision making with android's runtime permission dialogs using in-context surveys. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pages 195–210, Santa Clara, CA, 2017. USENIX Association.

[4] C.-C. Chang and C.-J. Lin. Libsvm – a library for support vector machines. http://www.csie.ntu.edu.tw/~cjlin/libsvm/. Accessed: September 11, 2017.

[5] G. Developer. Requesting permissions at run time. https://developer.android.com/training/permissions/ requesting.html. Accessed: September 16, 2017.

[6] S. Egelman, A. P. Felt, and D. Wagner. Choice architecture and smartphone privacy: There's a price for that. In *The 2012 Workshop on the Economics of Information Security (WEIS)*, 2012.

[7] Z. Fang, W. Han, D. Li, Z. Guo, D. Guo, X. S. Wang, Z. Qian, and H. Chen. revdroid: Code analysis of the side effects after dynamic permission revocation of android apps. In *Proceedings of the 11th ACM Asia Conference on Computer and Communications Security (ASIACCS 2016)*, Xi'an, China, 2016. ACM.

[8] A. P. Felt, S. Egelman, M. Finifter, D. Akhawe, and D. Wagner. How to ask for permission. In *Proceedings of the 7th USENIX conference on Hot Topics in Security*, HotSec'12, pages 7–7, Berkeley, CA, USA, 2012. USENIX Association.

[9] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: user attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, New York, NY, USA, 2012. ACM.

[10] Google. Dangerous permissions. https://developer.android. com/guide/topics/permissions/requesting.html# normal-dangerous. Accessed: August 17, 2017.

[11] R. Larson and M. Csikszentmihalyi. New directions for naturalistic methods in the behavioral sciences. In H. Reis, editor, *The Experience Sampling Method*, pages 41–56. Jossey-Bass, San Francisco, 1983.

[12] J. Lin, N. Sadeh, S. Amini, J. Lindqvist, J. I. Hong, and J. Zhang. Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing. In

Proceedings of the 2012 ACM Conference on Ubiquitous Computing, UbiComp '12, pages 501–510, New York, NY, USA, 2012. ACM.

[13] B. Liu, M. S. Andersen, F. Schaub, H. Almuhimedi, S. A. Zhang, N. Sadeh, Y. Agarwal, and A. Acquisti. Follow my recommendations: A personalized assistant for mobile app permissions. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, 2016.

[14] H. Nissenbaum. *Privacy in context: Technology, policy, and the integrity of social life*. Stanford University Press, 2009.

[15] Stanford Design School. Extreme users. https:// dschool-old.stanford.edu/wp-content/themes/ dschool/method-cards/extreme-users.pdf.

[16] J. Tan, K. Nguyen, M. Theodorides, H. Negron-Arroyo, C. Thompson, S. Egelman, and D. Wagner. The effect of developer-specified explanations for permission requests on smartphone user behavior. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2014.

[17] L. Tsai, P. Wijesekera, J. Reardon, I. Reyes, S. Egelman, D. Wagner, N. Good, and J.-W. Chen. Turtle guard: Helping android users apply contextual privacy preferences. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pages 145–162, Santa Clara, CA, 2017. USENIX Association.

[18] P. Wijesekera, A. Baokar, A. Hosseini, S. Egelman, D. Wagner, and Beznosov. Android permissions remystified: A field study on contextual integrity. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 499–514, Washington, D.C., Aug. 2015. USENIX Association.

[19] P. Wijesekera, A. Baokar, L. Tsai, J. Reardon, S. Egelman, D. Wagner, and Beznosov. The feasability of dynamically granted permissions:aligning mobile privacy with user preferences. In *Proceedings of the 2017 IEEE Symposium on Security and Privacy*, Oakland '17. IEEE Computer Society, 2017. To appear.

Appendix A Android Permissions Remystified: A Study on Contextual Integrity

Android Permissions Remystified: A Field Study on Contextual Integrity

Primal Wijesekera[1], Arjun Baokar[2], Ashkan Hosseini[2], Serge Egelman[2],
David Wagner[2], and Konstantin Beznosov[1]
[1]*University of British Columbia, Vancouver, Canada,*
{primal,beznosov}@ece.ubc.ca
[2]*University of California, Berkeley, Berkeley, USA,*
{arjunbaokar,ashkan}@berkeley.edu, {egelman,daw}@cs.berkeley.edu

Abstract

We instrumented the Android platform to collect data regarding how often and under what circumstances smartphone applications access protected resources regulated by permissions. We performed a 36-person field study to explore the notion of "contextual integrity," i.e., how often applications access protected resources when users are not expecting it. Based on our collection of 27M data points and exit interviews with participants, we examine the situations in which users would like the ability to deny applications access to protected resources. At least 80% of our participants would have preferred to prevent at least one permission request, and overall, they stated a desire to block over a third of all requests. Our findings pave the way for future systems to automatically determine the situations in which users would want to be confronted with security decisions.

1 Introduction

Mobile platform permission models regulate how applications access certain resources, such as users' personal information or sensor data (e.g., camera, GPS, etc.). For instance, previous versions of Android prompt the user during application installation with a list of all the permissions that the application may use in the future; if the user is uncomfortable granting any of these requests, her only option is to discontinue installation [3]. On iOS and Android M, the user is prompted at runtime the first time an application requests any of a handful of data types, such as location, address book contacts, or photos [34].

Research has shown that few people read the Android install-time permission requests and even fewer comprehend them [16]. Another problem is habituation: on average, Android applications present the user with four permission requests during the installation process [13]. While iOS users are likely to see fewer permission requests than Android users, because there are fewer possible permissions and they are only displayed the first time the data is actually requested, it is not clear whether or not users are being prompted about access to data that they actually find concerning, or whether they would approve of subsequent requests [15].

Nissenbaum posited that the reason why most privacy models fail to predict violations is that they fail to consider contextual integrity [32]. That is, privacy violations occur when personal information is used in ways that defy users' expectations. We believe that this notion of "privacy as contextual integrity" can be applied to smartphone permission systems to yield more effective permissions by only prompting users when an application's access to sensitive data is likely to defy expectations. As a first step down this path, we examined how applications are currently accessing this data and then examined whether or not it complied with users' expectations.

We modified Android to log whenever an application accessed a permission-protected resource and then gave these modified smartphones to 36 participants who used them as their primary phones for one week. The purpose of this was to perform dynamic analysis to determine how often various applications are actually accessing protected resources under realistic circumstances. Afterwards, subjects returned the phones to our laboratory and completed exit surveys. We showed them various instances over the past week where applications had accessed certain types of data and asked whether those instances were expected, and whether they would have wanted to deny access. Participants wanted to block a third of the requests. Their decisions were governed primarily by two factors: whether they had privacy concerns surrounding the specific data type and whether they understood why the application needed it.

We contribute the following:

- To our knowledge, we performed the first field study to quantify the permission usage by third-party applications under realistic circumstances.

- We show that our participants wanted to block access to protected resources a third of the time. This suggests that some requests should be granted by runtime consent dialogs, rather than Android's previous all-or-nothing install-time approval approach.
- We show that the visibility of the requesting application and the frequency at which requests occur are two important factors which need to be taken into account in designing a runtime consent platform.

2 Related Work

While users are required to approve Android application permission requests during installation, most do not pay attention and fewer comprehend these requests [16, 26]. In fact, even developers are not fully knowledgeable about permissions [40], and are given a lot of freedom when posting an application to the Google Play Store [7]. Applications often do not follow the principle of least privilege, intentionally or unintentionally [44]. Other work has suggested improving the Android per- mission model with better definitions and hierarchical breakdowns [8]. Some researchers have experimented with adding fine-grained access control to the Android model [11]. Providing users with more privacy information and personal examples has been shown to help users in choosing applications with fewer permissions [21,27].

Previous work has examined the overuse of permissions by applications [13, 20], and attempted to identify malicious applications through their permission requests [36] or through natural language processing of application descriptions [35]. Researchers have also developed static analysis tools to analyze Android permission specifications [6, 9, 13]. Our work complements this static analysis by applying dynamic analysis to permission us- age. Other researchers have applied dynamic analysis to native (non-Java) APIs among third-party mobile markets [39]; we apply it to the Java APIs available to developers in the Google Play Store.

Researchers examined user privacy expectations surrounding application permissions, and found that users were often surprised by the abilities of background applications to collect data [25, 42]. Their level of con- cern varied from annoyance to seeking retribution when presented with possible risks associated with permissions [15]. Some studies employed crowdsourcing to create a privacy model based on user expectations [30].

Researchers have designed systems to track or reduce privacy violations by recommending applications based on users' security concerns [2, 12, 19, 24, 28, 46–48]. Other tools dynamically block runtime permission requests [37]. Enck et al. found that a considerable number of applications transmitted location or other user data to third parties without requiring user consent [12]. Hornyack et al.'s AppFence system gave users the ability to deny data to applications or substitute fake data [24]. However, this broke application functionality for one-third of the applications tested.

Reducing the number of security decisions a user must make is likely to decrease habituation, and therefore, it is critical to identify *which* security decisions users should be asked to make. Based on this theory, Felt et al. created a decision tree to aid platform designers in determining the most appropriate permission-granting mechanism for a given resource (e.g., access to benign resources should be granted automatically, whereas access to dangerous resources should require approval) [14]. They concluded that the majority of Android permissions can be automatically granted, but 16% (corresponding to the 12 permissions in Table 1) should be granted via runtime dialogs.

Nissenbaum's theory of contextual integrity can help us to analyze "the appropriateness of a flow" in the context of permissions granted to Android applications [32]. There is ambiguity in defining when an application actually needs access to user data to run properly. It is quite easy to see why a location-sharing application would need access to GPS data, whereas that same request coming from a game like Angry Birds is less obvious. "Contextual integrity is preserved if information flows according to contextual norms" [32], however, the lack of thorough documentation on the Android permission model makes it easier for programmers to neglect these norms, whether intentionally or accidentally [38]. Deciding on whether an application is violating users' privacy can be quite complicated since "the scope of privacy is wide-ranging" [32]. To that end, we performed dynamic analysis to measure how often (and under what circumstances) applications were accessing protected resources, whether this complied with users' expectations, as well as how often they might be prompted if we adopt Felt et al.'s proposal to require runtime user confirmation before accessing a subset of these resources [14]. Finally, we show how it is possible to develop a classifier to automatically determine whether or not to prompt the user based on varying contextual factors.

3 Methodology

Our long-term research goal is to minimize habituation by only confronting users with *necessary* security decisions and avoiding showing them permission requests that are either expected, reversible, or unconcerning. Selecting which permissions to ask about requires understanding how often users would be confronted with each type of request (to assess the risk of habituation) and user reactions to these requests (to assess the benefit to users). In this study, we explored the problem space in two parts:

we instrumented Android so that we could collect actual usage data to understand how often access to various protected resources is requested by applications in practice, and then we surveyed our participants to understand the requests that they would not have granted, if given the option. This field study involved 36 participants over the course of one week of normal smartphone usage. In this section, we describe the log data that we collected, our recruitment procedure, and then our exit survey.

3.1 Tracking Access to Sensitive Data

In Android, when applications attempt to access protected resources (e.g., personal information, sensor data, etc.) at runtime, the operating system checks to see whether or not the requesting application was previously granted access during installation. We modified the Android platform to add a logging framework so that we could determine every time one of these resources was accessed by an application at runtime. Because our target device was a Samsung Nexus S smartphone, we modified Android 4.1.1 (Jellybean), which was the newest version of Android supported by our hardware.

3.1.1 Data Collection Architecture

Our goal was to collect as much data as possible about each applications' access to protected resources, while minimizing our impact on system performance. Our data collection framework consisted of two main components: a series of "producers" that hooked various Android API calls and a "consumer" embedded in the main Android framework service that wrote the data to a log file and uploaded it to our collection server.

We logged three kinds of permission requests. First, we logged function calls checked by `checkPermission()` in the Android `Context` implementation. Instrumenting the `Context` implementation, instead of the `ActivityManagerService` or `PackageManager`, allowed us to also log the function name invoked by the user-space application. Next, we logged access to the `ContentProvider` class, which verifies the read and write permissions of an application prior to it accessing structured data (e.g., contacts or calendars) [5]. Finally, we tracked permission checks during `Intent` transmission by instrumenting the `ActivityManagerService` and `BroadcastQueue`. `Intents` allow an application to pass messages to another application when an activity is to be performed in that other application (e.g., opening a URL in the web browser) [4].

We created a component called `Producer` that fetches the data from the above instrumented points and sends it back to the `Consumer`, which is responsible for logging everything reported. `Producers` are scattered across the Android Platform, since permission checks occur in multiple places. The `Producer` that logged the most data was in `system_server` and recorded direct function calls to Android's Java API. For a majority of privileged function calls, when a user application invokes the function, it sends the request to `system server` via `Binder`. `Binder` is the most prominent IPC mechanism implemented to communicate with the Android Platform (whereas `Intents` communicate between applications). For requests that do not make IPC calls to the `system_server`, a `Producer` is placed in the user application context (e.g., in the case of `ContentProviders`).

The `Consumer` class is responsible for logging data produced by each `Producer`. Additionally, the `Consumer` also stores contextual information, which we describe in Section 3.1.2. The `Consumer` syncs data with the filesystem periodically to minimize impact on system performance. All log data is written to the internal storage of the device because the Android kernel is not allowed to write to external storage for security reasons. Although this protects our data from curious or careless users, it also limits our storage capacity. Thus, we compressed the log files once every two hours and upload them to our collection servers whenever the phone had an active Internet connection (the average uploaded and zipped log file was around 108KB and contained 9,000 events).

Due to the high volume of permission checks we encountered and our goal of keeping system performance at acceptable levels, we added rate-limiting logic to the `Consumer`. Specifically, if it has logged permission checks for a particular application/permission combination more than 10,000 times, it examines whether it did so while exceeding an average rate of 1 permission check every 2 seconds. If so, the `Consumer` will only record 10% of all future requests for this application/permission combination. When this rate-limiting is enabled, the `Consumer` tracks these application/permission combinations and updates all the `Producers` so that they start dropping these log entries. Finally, the `Consumer` makes a note of whenever this occurs so that we can extrapolate the true number of permission checks that occurred.

3.1.2 Data Collection

We hooked the permission-checking APIs so that every time the system checked whether an application had been granted a particular permission, we logged the name of the permission, the name of the application, and the API method that resulted in the check. In addition to timestamps, we collected the following contextual data:

- **Visibility**—We categorized whether the requesting application was visible to the user, using four categories: running (a) as a service with no user interaction; (b) as a service, but with user interaction via

notifications or sounds; (c) as a foreground process, but in the background due to multitasking; or (d) as a foreground process with direct user interaction.
- **Screen Status**—Whether the screen was on/off.
- **Connectivity**—The phone's WiFi connection state.
- **Location**—The user's last known coordinates. In order to preserve battery life, we collected cached location data, rather than directly querying the GPS.
- **View**—The UI elements in the requesting application that were exposed to the user at the time that a protected resource was accessed. Specifically, since the UI is built from an XML file, we recorded the name of the screen as defined in the DOM.
- **History**—A list of applications with which the user interacted prior to the requesting application.
- **Path**—When access to a `ContentProvider` object was requested, the path to the specific content.

Felt et al. proposed granting most Android permissions without *a priori* user approval and granting 12 permissions (Table 1) at runtime so that users have contextual information to infer why the data might be needed [14]. The idea is that, if the user is asked to grant a permission while using an application, she may have some understanding of why the application needs that permission based on what she was doing. We initially wanted to perform experience sampling by probabilistically questioning participants whenever any of these 12 permissions were checked [29]. Our goal was to survey participants about whether access to these resources was expected and whether it should proceed, but we were concerned that this would prime them to the security focus of our experiment, biasing their subsequent behaviors. Instead, we instrumented the phones to probabilistically take screenshots of what participants were doing when these 12 permissions were checked so that we could ask them about it during the exit survey. We used reservoir sampling to minimize storage and performance impacts, while also ensuring that the screenshots covered a broad set of applications and permissions [43].

Figure 1 shows a screenshot captured during the study along with its corresponding log entry. The user was playing the Solitaire game while Spotify requested a WiFi scan. Since this permission was of interest (Table 1), our instrumentation took a screenshot. Since Spotify was not the application the participant was interacting with, its visibility was set to *false*. The history shows that prior to Spotify calling `getScanResults()`, the user had viewed Solitaire, the call screen, the launcher, and the list of MMS conversations.

Permission Type	Activity
WRITE_SYNC_SETTINGS	Change application sync settings when the user is roaming
ACCESS_WIFI_STATE	View nearby SSIDs
INTERNET	Access Internet when roaming
NFC	Communicate via NFC
READ_HISTORY_BOOKMARKS	Read users' browser history
ACCESS_FINE_LOCATION	Read GPS location
ACCESS_COARSE_LOCATION	Read network-inferred location (i.e., cell tower and/or WiFi)
LOCATION_HARDWARE	Directly access GPS data
READ_CALL_LOG	Read call history
ADD_VOICEMAIL	Read call history
READ_SMS	Read sent/received/draft SMS
SEND_SMS	Send SMS

Table 1: The 12 permissions that Felt et al. recommend be granted via runtime dialogs [14]. We randomly took screenshots when these permissions were requested by applications, and we asked about them in our exit survey.

3.2 Recruitment

We placed an online recruitment advertisement on Craigslist in October of 2014, under the "et cetera jobs" section.[1] The title of the advertisement was "Research Study on Android Smartphones," and it stated that the study was about how people interact with their smartphones. We made no mention of security or privacy. Those interested in participating were directed to an online consent form. Upon agreeing to the consent form, potential participants were directed to a screening application in the Google Play store. The screening application asked for information about each potential participant's age, gender, smartphone make and model. It also collected data on their phones' internal memory size and the installed applications. We screened out applicants who were under 18 years of age or used providers other than T-Mobile, since our experimental phones could not attain 3G speeds on other providers. We collected data on participants' installed applications so that we could preinstall free applications prior to them visiting our laboratory. (We copied paid applications from their phones, since we could not download those ahead of time.)

We contacted participants who met our screening requirements to schedule a time to do the initial setup. Overall, 48 people showed up to our laboratory, and of those, 40 qualified (8 were rejected because our screening application did not distinguish some Metro PCS users

[1] Approved by the UC Berkeley IRB under protocol #2013-02-4992

(a) Screenshot

Name	Log Data
Type	API_FUNC
Permission	ACCESS_WIFI_STATE
App Name	com.spotify.music
Timestamp	1412888326273
API Function	getScanResults()
Visibility	FALSE
Screen Status	SCREEN ON
Connectivity	NOT_CONNECTED
Location	Lat 37.XXX Long -122.XXX - 1412538686641 (Time it was updated)
View	com.mobilityware.solitaire/.Solitaire
History	com.android.phone/.InCallScreen com.android.launcher/com.android.-launcher2.Launcher com.android.mms/ConversationList

(b) Corresponding log entry

Figure 1: Screenshot (a) and corresponding log entry (b) captured during the experiment.

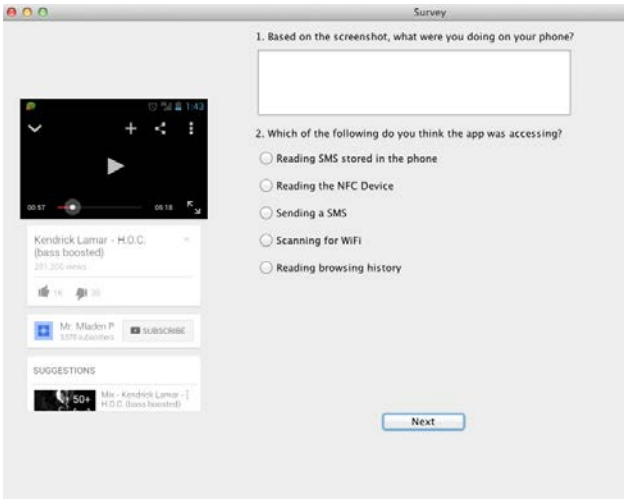
from T-Mobile users). In the email, we noted that due to the space constraints of our experimental phones, we might not be able to install all the applications on their existing phones, and therefore they needed to make a note of the ones that they planned to use that week. The initial setup took roughly 30 minutes and involved transferring their SIM cards, helping them set up their Google and other accounts, and making sure they had all the applications they needed. We compensated each participant with a $35 gift card for showing up at the setup session. Out of 40 people who were given phones, 2 did not return them, and 2 did not regularly use them during the study period. Of our 36 remaining participants who used the phones regularly, 19 were male and 17 were female; ages ranged from 20 to 63 years old ($\mu = 32$, $s = 11$).

After the initial setup session, participants used the experimental phones for one week in lieu of their normal phones. They were allowed to install and uninstall applications, and we instructed them to use these phones as they would their normal phones. Our logging framework kept track of every protected resource accessed by a user-level application along with the previously-mentioned contextual data. Due to storage constraints on the devices, our software uploaded log files to our server every two hours. However, to preserve participants' privacy, screenshots remained on the phones during the course of the week. At the end of the week, each participant returned to our laboratory, completed an exit survey, returned the phone, and then received an additional $100 gift card (i.e., slightly more than the value of the phone).

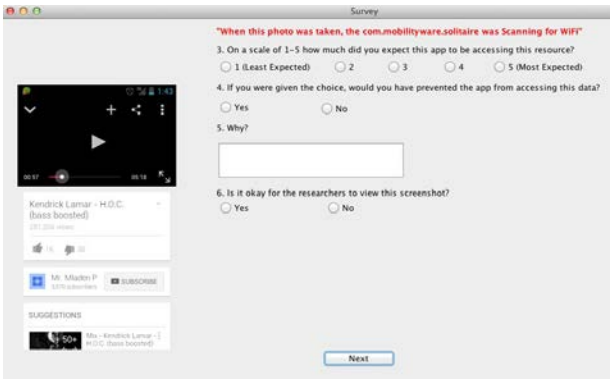
3.3 Exit Survey

When participants returned to our laboratory, they completed an exit survey. The exit survey software ran on a laptop in a private room so that it could ask questions about what they were doing on their phones during the course of the week without raising privacy concerns. We did not view their screenshots until participants gave us permission. The survey had three components:

- **Screenshots**—Our software displayed a screenshot taken after one of the 12 resources in Table 1 was accessed. Next to the screenshot (Figure 2a), we asked participants what they were doing on the phone when the screenshot was taken (open-ended). We also asked them to indicate which of several actions they believed the application was performing, chosen from a multiple-choice list of permissions presented in plain language (e.g., "reading browser history," "sending a SMS," etc.). After answering these questions, they proceeded to a second page of questions (Figure 2b). We informed participants at the top of this page of the resource that the application had accessed when the screenshot was taken, and asked them to indicate how much they expected this (5-point Likert scale). Next, we asked, "if you were given the choice, would you have prevented the app from accessing this data," and to explain why or why not. Finally, we asked for permission to view the screenshot. This phase of the exit survey was repeated for 10-15 different screenshots per participant, based on the number of screenshots saved by our reservoir sampling algorithm.
- **Locked Screens**—The second part of our survey involved questions about the same protected resources, though accessed while device screens were off (i.e., participants were not using their phones). Because there were no contextual cues (i.e., screenshots), we outright told participants which applications were accessing which resources and asked them multiple choice questions about whether they wanted to prevent this and the degree to which these



(a) On the first screen, participants answered questions to establish awareness of the permission request based on the screenshot.



(b) On the second screen, they saw the resource accessed, stated whether it was expected, and whether it should have been blocked.

Figure 2: Exit Survey Interface

behaviors were expected. They answered these questions for up to 10 requests, similarly chosen by our reservoir sampling algorithm to yield a breadth of application/permission combinations.

- **Personal Privacy Preferences**—Finally, in order to correlate survey responses with privacy preferences, participants completed two privacy scales. Because of the numerous reliability problems with the Westin index [45], we computed the average of both Buchanan et al.'s Privacy Concerns Scale (PCS) [10] and Malhotra et al.'s Internet Users' Information Privacy Concerns (IUIPC) scale [31].

After participants completed the exit survey, we re-entered the room, answered any remaining questions, and then assisted them in transferring their SIM cards back into their personal phones. Finally, we compensated each participant with a $100 gift card.

Three researchers independently coded 423 responses to the open-ended question in the screenshot portion of the survey. The number of responses per participant varied, as they were randomly selected based on the number of screenshots taken: participants who used their phones more heavily had more screenshots, and thus answered more questions. Prior to meeting to achieve consensus, the three coders disagreed on 42 responses, which resulted in an inter-rater agreement of 90%. Taking into account the 9 possible codings for each response, Fleiss' kappa yielded 0.61, indicating substantial agreement.

4 Application Behaviors

Over the week-long period, we logged 27M application requests to protected resources governed by Android permissions. This translates to over 100,000 requests per user/day. In this section, we quantify the circumstances under which these resources were accessed. We focus on the rate at which resources were accessed when participants were not actively using those applications (i.e., situations likely to defy users' expectations), access to certain resources with particularly high frequency, and the impact of replacing certain requests with runtime confirmation dialogs (as per Felt et al.'s suggestion [14]).

4.1 Invisible Permission Requests

In many cases, it is entirely expected that an application might make frequent requests to resources protected by permissions. For instance, the INTERNET permission is used every time an application needs to open a socket, ACCESS_FINE_LOCATION is used every time the user's location is checked by a mapping application, and so on. However, in these cases, one expects users to have certain contextual cues to help them understand that these applications are running and making these requests. Based on our log data, most requests occurred while participants were not actually interacting with those applications, nor did they have any cues to indicate that the applications were even running. When resources are accessed, applications can be in five different states, with regard to their visibility to users:

1. **Visible foreground application (12.04%)**: the user is using the application requesting the resource.
2. **Invisible background application (0.70%)**: due to multitasking, the application is in the background.
3. **Visible background service (12.86%)**: the application is a background service, but the user may be aware of its presence due to other cues (e.g., it is playing music or is present in the notification bar).
4. **Invisible background service (14.40%)**: the application is a background service without visibility.
5. **Screen off (60.00%)**: the application is running, but the phone screen is off because it is not in use.

Permission	Requests
ACCESS_NETWORK_STATE	31,206
WAKE_LOCK	23,816
ACCESS_FINE_LOCATION	5,652
GET_ACCOUNTS	3,411
ACCESS_WIFI_STATE	1,826
UPDATE_DEVICE_STATS	1,426
ACCESS_COARSE_LOCATION	1,277
AUTHENTICATE_ACCOUNTS	644
READ_SYNC_SETTINGS	426
INTERNET	416

Table 2: The most frequently requested permissions by applications with zero visibility to the user.

Application	Requests
Facebook	36,346
Google Location Reporting	31,747
Facebook Messenger	22,008
Taptu DJ	10,662
Google Maps	5,483
Google Gapps	4,472
Foursquare	3,527
Yahoo Weather	2,659
Devexpert Weather	2,567
Tile Game(Umoni)	2,239

Table 3: The applications making the most permission requests while running invisibly to the user.

Combining the 3.3M (12.04% of 27M) requests that were granted when the user was actively using the application (Category 1) with the 3.5M (12.86% of 27M) requests that were granted when the user had other contextual cues to indicate that the application was running (Category 3), we can see that fewer than one quarter of all permission requests (24.90% of 27M) occurred when the user had clear indications that those applications were running. This suggests that during the vast majority of the time, access to protected resources occurs opaquely to users. We focus on these 20.3M "invisible" requests (75.10% of 27M) in the remainder of this subsection.

Harbach et al. found that users' phone screens are off 94% of the time on average [22]. We observed that 60% of permission requests occurred while participants' phone screens were off, which suggests that permission requests occurred less frequently than when participants were using their phones. At the same time, certain applications made more requests when participants were not using their phones: "Brave Frontier Service," "Microsoft Sky Drive," and "Tile game by UMoni." Our study collected data on over 300 applications, and therefore it is possible that with a larger sample size, we would observe other applications engaging in this behavior. All of the aforementioned applications primarily requested ACCESS_WIFI_STATE and INTERNET. While a definitive explanation for this behavior requires examining source code or the call stacks of these applications, we hypothesize that they were continuously updating local data from remote servers. For instance, Sky Drive may have been updating documents, whereas the other two applications may have been checking the status of multiplayer games.

Table 2 shows the most frequently requested permissions from applications running invisibly to the user (i.e., Categories 2, 4, and 5); Table 3 shows the applications responsible for these requests (Appendix A lists the permissions requested by these applications). We normalized the numbers to show requests per user/day. ACCESS_NETWORK_STATE was most frequently requested, averaging 31,206 times per user/day—roughly once every 3 seconds. This is due to applications constantly checking for Internet connectivity. However, the 5,562 requests/day to ACCESS_FINE_LOCATION and 1,277 requests/day to ACCESS_COARSE_LOCATION are more concerning, as this could enable detailed tracking of the user's movement throughout the day. Similarly, a user's location can be inferred by using ACCESS_WIFI_STATE to get data on nearby WiFi SSIDs.

Contextual integrity means ensuring that information flows are appropriate, as determined by the user. Thus, users need the ability to see information flows. Current mobile platforms have done some work to let the user know about location tracking. For instance, recent versions of Android allow users to see which applications have used location data recently. While attribution is a positive step towards contextual integrity, attribution is most beneficial for actions that are reversible, whereas the disclosure of location information is not something that can be undone [14]. We observed that fewer than 1% of location requests were made when the applications were visible to the user or resulted in the displaying of a GPS notification icon. Given that Thompson et al. showed that most users do not understand that applications running in the background may have the same abilities as applications running in the foreground [42], it is likely that in the vast majority of cases, users do not know when their locations are being disclosed.

This low visibility rate is because Android only shows a notification icon when the GPS sensor is accessed, while offering alternative ways of inferring location. In 66.1% of applications' location requests, they directly queried the `TelephonyManager`, which can be used to determine location via cellular tower information. In 33.3% of the cases, applications requested the SSIDs of nearby WiFi networks. In the remaining 0.6% of cases, applica-

tions accessed location information using one of three built-in location providers: GPS, network, or passive. Applications accessed the GPS location provider only 6% of the time (which displayed a GPS notification). In the other 94% of the time, 13% queried the network provider (i.e., approximate location based on nearby cellular towers and WiFi SSIDs) and 81% queried the passive location provider. The passive location provider caches prior requests made to either the GPS or network providers. Thus, across all requests for location data, the GPS notification icon appeared 0.04% of the time.

While the alternatives to querying the GPS are less accurate, users are still surprised by their accuracy [17]. This suggests a serious violation of contextual integrity, since users likely have no idea their locations are being requested in the vast majority of cases. Thus, runtime notifications for location tracking need to be improved [18].

Apart from these invisible location requests, we also observed applications reading stored SMS messages (125 times per user/day), reading browser history (5 times per user/day), and accessing the camera (once per user/day). Though the use of these permissions does not necessarily lead to privacy violations, users have no contextual cues to understand that these requests are occurring.

4.2 High Frequency Requests

Some permission requests occurred so frequently that a few applications (i.e., Facebook, Facebook Messenger, Google Location Reporting, Google Maps, Farm Heroes Saga) had to be rate limited in our log files (see Section 3.1.1), so that the logs would not fill up users' remaining storage or incur performance overhead. Table 4 shows the complete list of application/permission combinations that exceeded the threshold. For instance, the most frequent requests came from Facebook requesting ACCESS_NETWORK_STATE with an average interval of 213.88 ms (i.e., almost 5 times per second).

With the exception of Google's applications, all rate-limited applications made excessive requests for the connectivity state. We hypothesize that once these applications lose connectivity, they continuously poll the system until it is regained. Their use of the `getActiveNetworkInfo()` method results in permission checks and returns `NetworkInfo` objects, which allow them to determine connection state (e.g., connected, disconnected, etc.) and type (e.g., WiFi, Bluetooth, cellular, etc.). Thus, these requests do not appear to be leaking sensitive information *per se*, but their frequency may have adverse effects on performance and battery life. It is possible that using the `ConnectivityManager`'s `NetworkCallback` method may be able to fulfill this need with far fewer permission checks.

Application / Permission	Peak (ms)	Avg. (ms)
com.facebook.katana ACCESS_NETWORK_STATE	213.88	956.97
com.facebook.orca ACCESS_NETWORK_STATE	334.78	1146.05
com.google.android.apps.maps ACCESS_NETWORK_STATE	247.89	624.61
com.google.process.gapps AUTHENTICATE_ACCOUNTS	315.31	315.31
com.google.process.gapps WAKE_LOCK	898.94	1400.20
com.google.process.location WAKE_LOCK	176.11	991.46
com.google.process.location ACCESS_FINE_LOCATION	1387.26	1387.26
com.google.process.location GET_ACCOUNTS	373.41	1878.88
com.google.process.location ACCESS_WIFI_STATE	1901.91	1901.91
com.king.farmheroessaga ACCESS_NETWORK_STATE	284.02	731.27
com.pandora.android ACCESS_NETWORK_STATE	541.37	541.37
com.taptu.streams ACCESS_NETWORK_STATE	1746.36	1746.36

Table 4: The application/permission combinations that needed to be rate limited during the study. The last two columns show the fastest interval recorded and the average of all the intervals recorded before rate-limiting.

4.3 Frequency of Data Exposure

Felt et al. posited that while most permissions can be granted automatically in order to not habituate users to relatively benign risks, certain requests should require runtime consent [14]. They advocated using runtime dialogs before the following actions should proceed:

1. Reading location information (e.g., using conventional location APIs, scanning WiFi SSIDs, etc.).
2. Reading the user's web browser history.
3. Reading saved SMS messages.
4. Sending SMS messages that incur charges, or inappropriately spamming the user's contact list.

These four actions are governed by the 12 Android permissions listed in Table 1. Of the 300 applications that we observed during the experiment, 91 (30.3%) performed one of these actions. On average, these permissions were requested 213 times per hour/user—roughly every 20 seconds. However, permission checks occur under a variety of circumstances, only a subset of which expose sensitive resources. As a result, platform develop-

Resource	Visible		Invisible		Total	
	Data Exposed	Requests	Data Exposed	Requests	Data Exposed	Requests
Location	758	2,205	3,881	8,755	4,639	10,960
Read SMS data	378	486	72	125	450	611
Sending SMS	7	7	1	1	8	8
Browser History	12	14	2	5	14	19
Total	1,155	2,712	3,956	8,886	5,111	11,598

Table 5: The sensitive permission requests (per user/day) when requesting applications were visible/invisible to users. "Data exposed" reflects the subset of permission-protected requests that resulted in sensitive data being accessed.

ers may decide to only show runtime warnings to users when protected data is read or modified. Thus, we attempted to quantify the frequency with which permission checks actually result in access to sensitive resources for each of these four categories. Table 5 shows the number of requests seen per user/day under each of these four categories, separating the instances in which sensitive data was exposed from the total permission requests observed. Unlike Section 4.1, we include "visible" permission requests (i.e., those occurring while the user was actively using the application or had other contextual information to indicate it was running). We didn't observe any uses of NFC, READ_CALL_LOG, ADD_VOICEMAIL, accessing WRITE_SYNC_SETTINGS or INTERNET while roaming in our dataset.

Of the location permission checks, a majority were due to requests for location provider information (e.g., `getBestProvider()` returns the best location provider based on application requirements), or checking WiFi state (e.g., `getWifiState()` only reveals whether WiFi is enabled). Only a portion of the requests actually exposed participants' locations (e.g., `getLastKnownLocation()` or `getScanResults()` exposed SSIDs of nearby WiFi networks).

Although a majority of requests for the READ_SMS permission exposed content in the SMS store (e.g., `Query()` reads the contents of the SMS store), a considerable portion simply read information about the SMS store (e.g., `renewMmsConnectivity()` resets an applications' connection to the MMS store). An exception to this is the use of SEND_SMS, which resulted in the transmission of an SMS message every time the permission was requested.

Regarding browser history, both accessing visited URLs (`getAllVisitedUrls()`) and reorganizing bookmark folders (`addFolderToCurrent()`) result in the same permission being checked. However, the latter does not expose specific URLs to the invoking application.

Our analysis of the API calls indicated that on average, only half of all permission checks granted applications access to sensitive data. For instance, across both visible and invisible requests, 5,111 of the 11,598 (44.3%) permission checks involving the 12 permissions in Table 1 resulted in the exposure of sensitive data (Table 5).

While limiting runtime permission requests to only the cases in which protected resources are exposed will greatly decrease the number of user interruptions, the frequency with which these requests occur is still too great. Prompting the user on the first request is also not appropriate (e.g., à la iOS and Android M), because our data show that in the vast majority of cases, the user has no contextual cues to understand when protected resources are being accessed. Thus, a user may grant a request the first time an application asks, because it is appropriate in that instance, but then she may be surprised to find that the application continues to access that resource in other contexts (e.g., when the application is not actively used). As a result, a more intelligent method is needed to determine when a given permission request is likely to be deemed appropriate by the user.

5 User Expectations and Reactions

To identify when users might want to be prompted about permission requests, our exit survey focused on participants' reactions to the 12 permissions in Table 1, limiting the number of requests shown to each participant based on our reservoir sampling algorithm, which was designed to ask participants about a diverse set of permission/application combinations. We collected participants' reactions to 673 permission requests (≈19/participant). Of these, 423 included screenshots because participants were actively using their phones when the requests were made, whereas 250 permission requests were performed while device screens were off.[2] Of the former, 243 screenshots were taken while the requesting application was visible (Category 1 and 3 from Section 4.1), whereas 180 were taken while the application was invisible (Category 2 and 4 from Section 4.1). In this section, we describe the situations in which requests

[2]Our first 11 participants did not answer questions about permission requests occurring while not using their devices, and therefore the data only corresponds to our last 25 participants.

defied users' expectations. We present explanations for why participants wanted to block certain requests, the factors influencing those decisions, and how expectations changed when devices were not in use.

5.1 Reasons for Blocking

When viewing screenshots of what they were doing when an application requested a permission, 30 participants (80% of 36) stated that they would have preferred to block at least one request, whereas 6 stated a willingness to allow all requests, regardless of resource type or application. Across the entire study, participants wanted to block 35% of these 423 permission requests. When we asked participants to explain their rationales for these decisions, two main themes emerged: the request did not—in their minds—pertain to application functionality or it involved information they were uncomfortable sharing.

5.1.1 Relevance to Application Functionality

When prompted for the reason behind blocking a permission request, 19 (53% of 36) participants did not believe it was necessary for the application to perform its task. Of the 149 (35% of 423) requests that participants would have preferred to block, 79 (53%) were perceived as being irrelevant to the functionality of the application:

- *"It wasn't doing anything that needed my current location."* (P1)
- *"I don't understand why this app would do anything with SMS."* (P10)

Accordingly, functionality was the most common reason for wanting a permission request to proceed. Out of the 274 permissible requests, 195 (71% of 274) were perceived as necessary for the core functionality of the application, as noted by thirty-one (86% of 36) participants:

- *"Because it's a weather app and it needs to know where you are to give you weather information."*(P13)
- *"I think it needs to read the SMS to keep track of the chat conversation."*(P12)

Beyond being necessary for core functionality, participants wanted 10% (27 of 274) of requests to proceed because they offered convenience; 90% of these requests were for location data, and the majority of those applications were published under the Weather, Social, and Travel & Local categories in the Google Play store:

- *"It selects the closest stop to me so I don't have to scroll through the whole list."* (P0)
- *"This app should read my current location. I'd like for it to, so I won't have to manually enter in my zip code / area."* (P4)

Thus, requests were allowed when they were expected: when participants rated the extent to which each request was expected on a 5-point Likert scale, allowable requests averaged 3.2, whereas blocked requests averaged 2.3 (lower is less expected).

5.1.2 Privacy Concerns

Participants also wanted to deny permission requests that involved data that they considered sensitive, regardless of whether they believed the application actually needed the data to function. Nineteen (53% of 36) participants noted privacy as a concern while blocking a request, and of the 149 requests that participants wanted to block, 49 (32% of 149) requests were blocked for this reason:

- *"SMS messages are quite personal."* (P14)
- *"It is part of a personal conversation."* (P11)
- *"Pictures could be very private and I wouldn't like for anybody to have access."* (P16)

Conversely, 24 participants (66% of 36) wanted requests to proceed simply because they did not believe that the data involved was particularly sensitive; this reasoning accounted for 21% of the 274 allowable requests:

- *"I'm ok with my location being recorded, no concerns."* (P3)
- *"No personal info being shared."* (P29)

5.2 Influential Factors

Based on participants' responses to the 423 permission requests involving screenshots (i.e., requests occurring while they were actively using their phones), we quantitatively examined how various factors influenced their desire to block some of these requests.

Effects of Identifying Permissions on Blocking: In the exit survey, we asked participants to guess the permission an application was requesting, based on the screenshot of what they were doing at the time. The real answer was among four other incorrect answers. Of the 149 cases where participants wanted to block permission requests, they were only able to correctly state what permission was being requested 24% of the time; whereas when wanting a request to proceed, they correctly identified the requested permission 44% (120 of 274) of the time. However, Pearson's product-moment test on the average number of blocked requests per user and the average number of correct answers per user[3] did not yield a statistically significant correlation ($r=-0.171$, $p<0.317$).

Effects of Visibility on Expectations: We were particularly interested in exploring if permission requests originating from foreground applications (i.e., visible to the

[3]Both measures were normally distributed.

user) were more expected than ones from background applications. Of the 243 visible permission requests that we asked about in our exit survey, participants correctly identified the requested permission 44% of the time, and their average rating on our expectation scale was 3.4. On the other hand, participants correctly identified the resources accessed by background applications only 29% of the time (52 of 180), and their average rating on our expectation scale was 3.0. A Wilcoxon Signed-Rank test with continuity correction revealed a statistically significant difference in participants' expectations between these two groups ($V=441.5$, $p<0.001$).

Effects of Visibility on Blocking: Participants wanted to block 71 (29% of 243) permission requests originating from applications running in the foreground, whereas this increased by almost 50% when the applications were in the background invisible to them (43% of 180). We calculated the percentage of denials for each participant, for both visible and invisible requests. A Wilcoxon Signed-Rank test with continuity correction revealed a statistically significant difference ($V=58$, $p<0.001$).

Effects of Privacy Preferences on Blocking: Participants completed the Privacy Concerns Scale (PCS) [10] and the Internet Users' Information Privacy Concerns (IUIPC) scale [31]. A Spearman's rank test yielded no statistically significant correlation between their privacy preferences and their desire to block permission requests ($r = 0.156$, $p<0.364$).

Effects of Expectations on Blocking: We examined whether participants' expectations surrounding requests correlated with their desire to block them. For each participant, we calculated their average Likert scores for their expectations and the percentage of requests that they wanted to block. Pearson's product-moment test showed a statistically significant correlation ($r=-0.39$, $p<0.018$). The negative correlation shows that participants were more likely to deny unexpected requests.

5.3 User Inactivity and Resource Access

In the second part of the exit survey, participants answered questions about 10 resource requests that occurred when the screen was off (not in use). Overall, they were more likely to expect resource requests to occur when using their devices ($\mu = 3.26$ versus $\mu = 2.66$). They also stated a willingness to block almost half of the permission requests (49.6% of 250) when not in use, compared to a third of the requests that occurred when using their phones (35.2% of 423). However, neither of these differences was statistically significant.

6 Feasibility of Runtime Requests

Felt et al. posited that certain sensitive permissions (Table 1) should require runtime consent [14], but in Section 4.3 we showed that the frequencies with which applications are requesting these permissions make it impractical to prompt the user each time a request occurs. Instead, the major mobile platforms have shifted towards a model of prompting the user the first time an application requests access to certain resources: iOS does this for a selected set of resources, such as location and contacts, and Android M does this for "dangerous" permissions.

How many prompts would users see, if we added runtime prompts for the first use of these 12 permissions? We analyzed a scheme where a runtime prompt is displayed at most once for each unique triplet of (application, permission, application visibility), assuming the screen is on. With a naïve scheme, our study data indicates our participants would have seen an average of 34 runtime prompts (ranging from 13 to 77, $s=11$). As a refinement, we propose that the user should be prompted only if sensitive data will be exposed (Section 4.3), reducing the average number of prompts to 29.

Of these 29 prompts, 21 (72%) are related to location. Apple iOS already prompts users when an application accesses location for the first time, with no evidence of user habituation or annoyance. Focusing on the remaining prompts, we see that our policy would introduce an average of 8 new prompts per user: about 5 for reading SMS, 1 for sending SMS, and 2 for reading browser history. Our data covers only the first week of use, but as we only prompt on first use of a permission, we expect that the number of prompts would decline greatly in subsequent weeks, suggesting that this policy would likely not introduce significant risk of habituation or annoyance. Thus, our results suggest adding runtime prompts for reading SMS, sending SMS, and reading browser history would be useful given their sensitivity and low frequency.

Our data suggests that taking visibility into account is important. If we ignore visibility and prompted only once for each pair of (application, permission), users would have no way to select a different policy for when the application is visible or not visible. In contrast, "ask- on-first-use" for the triple (application, permission, visibility) gives users the option to vary their decision based on the visibility of the requesting application. We evaluated these two policies by analyzing the exit survey data (limited to situations where the screen was on) for cases where the same user was asked twice in the survey about situations with the same (application, permission) pair or the same (application, permission, visibility) triplet, to see whether the user's first decision to block or not matched their subsequent decisions. For the former pol-

icy, we saw only 51.3% agreement; for the latter, agreement increased to 83.5%. This suggests that the (application, permission, visibility) triplet captures many of the contextual factors that users care about, and thus it is reasonable to prompt only once per unique triplet.

A complicating factor is that applications can also run even when the user is not actively using the phone. In addition to the 29 prompts mentioned above, our data indicates applications would have triggered an average of 7 more prompts while the user was not actively using the phone: 6 for location and one for reading SMS. It is not clear how to handle prompts when the user is not available to respond to the prompt: attribution might be helpful, but further research is needed.

6.1 Modeling Users' Decisions

We constructed several statistical models to examine whether users' desire to block certain permission requests could be predicted using the contextual data that we collected. If such a relationship exists, a classifier could determine when to deny potentially unexpected permission requests without user intervention. Conversely, the classifier could be used to only prompt the user about questionable data requests. Thus, the response variable in our models is the user's choice of whether to block the given permission request. Our predictive variables consisted of the information that might be available at runtime: permission type (with the restriction that the invoked function exposes data), requesting application, and visibility of that application. We constructed several mixed effects binary logistic regression models to account for both inter-subject and intra-subject effects.

6.1.1 Model Selection

In our mixed effects models, permission types and the visibility of the requesting application were fixed effects, because all possible values for each variable existed in our data set. Visibility had two values: visible (the user is interacting with the application or has other contextual cues to know that it is running) and invisible. Permission types were categorized based on Table 5. The application name and the participant ID were included as random effects, because our survey data did not have an exhaustive list of all possible applications a user could run, and the participant has a non-systematic effect on the data.

Table 6 shows two goodness-of-fit metrics: the Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC). Lower values for AIC and BIC represent better fit. Table 6 shows the different parameters included in each model. We found no evidence of interaction effects and therefore did not include them. Visual inspection of residual plots of each model did not reveal obvious deviations from homoscedasticity or normality.

Predictors	AIC	BIC	Screen State
UserCode	490.60	498.69	Screen On
Application	545.98	554.07	Screen On
Application UserCode	491.86	503.99	Screen On
Permission Application UserCode	494.69	527.05	Screen On
Visibility Application UserCode	**481.65**	**497.83**	**Screen On**
Permission Visibility Application UserCode	484.23	520.64	Screen On
UserCode	245.13	252.25	Screen Off
Application	349.38	356.50	Screen Off
Application UserCode	238.84	249.52	Screen Off
Permission Application UserCode	**235.48**	**263.97**	**Screen Off**

Table 6: Goodness-of-fit metrics for various mixed effects logistic regression models on the exit survey data.

We initially included the phone's screen state as another variable. However, we found that creating two separate models based on the screen state resulted in better fit than using a single model that accounted for screen state as a fixed effect. When the screen was on, the best fit was a model including application visibility and appli- cation name, while controlling for subject effects. Here, fit improved once permission type was removed from the model, which shows that the decision to block a permission request was based on contextual factors: users do not categorically deny permission requests based solely on the type of resource being accessed (i.e., they also account for their trust in the application, as well as whether they happened to be actively using it). When the screen was off, however, the effect of permission type was relatively stronger. The strong subject effect in both models indicates that these decisions vary from one user to the next. As a result, any classifier developed to automatically decide whether to block a permission at runtime (or prompt the user) will need to be tailored to that particular user's needs.

6.1.2 Predicting User Reactions

Using these two models, we built two classifiers to make decisions about whether to block any of the sensitive permission requests listed in Table 5. We used our exit survey data as ground truth, and used 5-fold cross-validation to evaluate model accuracy.

We calculated the receiver operating characteristic (ROC) to capture the tradeoff between true-positive and false-positive rate. The quality of the classifier can be quantified with a single value by calculating the area under its ROC curve (AUC) [23]. The closer the AUC gets to 1.0, the better the classifier is. When screens were on, the AUC was 0.7, which is 40% better than the random baseline (0.5). When screens were off, the AUC was 0.8, which is 60% better than a random baseline.

7 Discussion

During the study, 80% of our participants deemed at least one permission request as inappropriate. This violates Nissenbaum's notion of "privacy as contextual integrity" because applications were performing actions that defied users' expectations [33]. Felt et al. posited that users may be able to better understand why permission requests are needed if some of these requests are granted via runtime consent dialogs, rather than Android's previous install-time notification approach [14]. By granting permissions at runtime, users will have additional contextual information; based on what they were doing at the time that resources are requested, they may have a better idea of why those resources are being requested.

We make two primary contributions that system designers can use to make more usable permissions systems. We show that the visibility of the requesting application and the frequency at which requests occur are two important factors in designing a runtime consent platform. Also, we show that "prompt-on-first-use" per triplet could be implemented for some sensitive permissions without risking user habituation or annoyance.

Based on the frequency with which runtime permissions are requested (Section 4), it is infeasible to prompt users every time. Doing so would overwhelm them and lead to habituation. At the same time, drawing user attention to the situations in which users are likely to be concerned will lead to greater control and awareness. Thus, the challenge is in acquiring their preferences by confronting them minimally and then automatically inferring *when* users are likely to find a permission request unexpected, and only prompting them in these cases. Our data suggests that participants' desires to block particular permissions were heavily influenced by two main factors: their understanding of the relevance of a permission request to the functionality of the requesting application and their individual privacy concerns.

Our models in Section 6.1 showed that individual characteristics greatly explain the variance between what different users deem appropriate, in terms of access to protected resources. While responses to privacy scales failed to explain these differences, this was not a surprise, as the disconnect between stated privacy preferences and behaviors is well-documented (e.g., [1]). This means that in order to accurately model user preferences, the system will need to learn what a specific user deems inappropriate over time. Thus, a feedback loop is likely needed: when devices are "new," users will be required to provide more input surrounding permission requests, and then based on their responses, they will see fewer requests in the future. Our data suggests that prompting once for each unique (application, permission, application visibility) triplet can serve as a practical mechanism in acquiring users' privacy preferences.

Beyond individual subject characteristics (i.e., personal preferences), participants based their decisions to block certain permission requests on the specific applications making the requests and whether they had contextual cues to indicate that the applications were running (and therefore needed the data to function). Future systems could take these factors into account when deciding whether or not to draw user attention to a particular request. For example, when an application that a user is not actively using requests access to a protected resource, she should be shown a runtime prompt. Our data indicates that, if the user decides to grant a request in this situation, then with probability 0.84 the same decision will hold in future situations where she is actively using that same application, and therefore a subsequent prompt may not be needed. At a minimum, platforms need to treat permission requests from background applications differently than those originating from foreground applications. Similarly, applications running in the background should use passive indicators to communicate when they are accessing particular resources. Platforms can also be designed to make decisions about whether or not access to resources should be granted based on whether contextual cues are present, or at its most basic, whether the device screen is even on.

Finally, we built our models and analyzed our data within the framework of what resources our participants *believed* were necessary for applications to correctly function. Obviously, their perceptions may have been incorrect: if they better understood why a particular resource was necessary, they may have been more permissive. Thus, it is incumbent on developers to adequately communicate why particular resources are needed, as this impacts user notions of contextual integrity. Yet, no mechanisms in Android exist for developers to do this as part of the permission-granting process. For example, one could imagine requiring metadata to be provided that explains how each requested resource will be used, and then automatically integrating this information into permission requests. Tan et al. examined a similar feature on iOS that allows developers to include free-form text in runtime

permission dialogs and observed that users were more likely to grant requests that included this text [41]. Thus, we believe that including succinct explanations in these requests would help preserve contextual integrity by promoting greater transparency.

In conclusion, we believe this study was instructive in showing the circumstances in which Android permission requests are made under real-world usage. While prior work has already identified some limitations of deployed mobile permissions systems, we believe our study can benefit system designers by demonstrating several ways in which contextual integrity can be improved, thereby empowering users to make better security decisions.

Acknowledgments

This work was supported by NSF grant CNS-1318680, by Intel through the ISTC for Secure Computing, and by the AFOSR under MURI award FA9550-12-1-0040.

References

[1] ACQUISTI, A., AND GROSSKLAGS, J. Privacy and rationality in individual decision making. *IEEE Security & Privacy* (January/February 2005), 24–30. `http://www.dtc.umn.edu/weis2004/acquisti.pdf`.

[2] ALMOHRI, H. M., YAO, D. D., AND KAFURA, D. Droidbarrier: Know what is executing on your android. In *Proc. of the 4th ACM Conf. on Data and Application Security and Privacy* (New York, NY, USA, 2014), CODASPY '14, ACM, pp. 257–264.

[3] ANDROID DEVELOPERS. System permissions. `http://developer.android.com/guide/topics/security/ permissions.html`. Accessed: November 11, 2014.

[4] ANDROID DEVELOPERS. Common Intents. `https://developer.android.com/guide/components/intents- common.html`, 2014. Accessed: November 12, 2014.

[5] ANDROID DEVELOPERS. Content Providers. `http://developer.android.com/guide/topics/providers/ content-providers.html`, 2014. Accessed: Nov. 12, 2014.

[6] AU, K. W. Y., ZHOU, Y. F., HUANG, Z., AND LIE, D. Pscout: Analyzing the android permission specification. In *Proc. of the 2012 ACM Conf. on Computer and Communications Security* (New York, NY, USA, 2012), CCS '12, ACM, pp. 217–228.

[7] BARRERA, D., CLARK, J., MCCARNEY, D., AND VAN OORSCHOT, P. C. Understanding and improving app installation security mechanisms through empirical analysis of android. In *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices* (New York, NY, USA, 2012), SPSM '12, ACM, pp. 81–92.

[8] BARRERA, D., KAYACIK, H. G. U. C., VAN OORSCHOT, P. C., AND SOMAYAJI, A. A methodology for empirical analysis of permission-based security models and its application to android. In *Proc. of the ACM Conf. on Comp. and Comm. Security* (New York, NY, USA, 2010), CCS '10, ACM, pp. 73–84.

[9] BODDEN, E. Easily instrumenting android applications for security purposes. In *Proc. of the ACM Conf. on Comp. and Comm. Sec.* (NY, NY, USA, 2013), CCS '13, ACM, pp. 1499–1502.

[10] BUCHANAN, T., PAINE, C., JOINSON, A. N., AND REIPS, U.-D. Development of measures of online privacy concern and protection for use on the internet. *Journal of the American Society for Information Science and Technology 58*, 2 (2007), 157–165.

[11] BUGIEL, S., HEUSER, S., AND SADEGHI, A.-R. Flexible and fine-grained mandatory access control on android for diverse security and privacy policies. In *Proc. of the 22nd USENIX Security Symposium* (Berkeley, CA, USA, 2013), SEC'13, USENIX Association, pp. 131–146.

[12] ENCK, W., GILBERT, P., CHUN, B.-G., COX, L. P., JUNG, J., MCDANIEL, P., AND SHETH, A. N. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2010), OSDI'10, USENIX Association, pp. 1–6.

[13] FELT, A. P., CHIN, E., HANNA, S., SONG, D., AND WAGNER, D. Android permissions demystified. In *Proc. of the ACM Conf. on Comp. and Comm. Sec.* (New York, NY, USA, 2011), CCS '11, ACM, pp. 627–638.

[14] FELT, A. P., EGELMAN, S., FINIFTER, M., AKHAWE, D., AND WAGNER, D. How to ask for permission. In *Proceedings of the 7th USENIX conference on Hot Topics in Security* (Berkeley, CA, USA, 2012), HotSec'12, USENIX Association, pp. 7–7.

[15] FELT, A. P., EGELMAN, S., AND WAGNER, D. I've got 99 problems, but vibration ain't one: a survey of smartphone users' concerns. In *Proc. of the 2nd ACM workshop on Security and Privacy in Smartphones and Mobile devices* (New York, NY, USA, 2012), SPSM '12, ACM, pp. 33–44.

[16] FELT, A. P., HA, E., EGELMAN, S., HANEY, A., CHIN, E., AND WAGNER, D. Android permissions: user attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security* (New York, NY, USA, 2012), SOUPS '12, ACM, pp. 3:1–3:14.

[17] FU, H., AND LINDQVIST, J. General area or approximate location?: How people understand location permissions. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society* (2014), ACM, pp. 117–120.

[18] FU, H., YANG, Y., SHINGTE, N., LINDQVIST, J., AND GRUTESER, M. A field study of run-time location access dis- closures on android smartphones. *Proc. USEC 14* (2014).

[19] GIBLER, C., CRUSSELL, J., ERICKSON, J., AND CHEN, H. Androidleaks: Automatically detecting potential privacy leaks in android applications on a large scale. In *Proc. of the 5th Intl. Conf. on Trust and Trustworthy Computing* (Berlin, Heidelberg, 2012), TRUST'12, Springer-Verlag, pp. 291–307.

[20] GORLA, A., TAVECCHIA, I., GROSS, F., AND ZELLER, A. Checking app behavior against app descriptions. In *Proceedings of the 36th International Conference on Software Engineering* (New York, NY, USA, 2014), ICSE 2014, ACM, pp. 1025–1035.

[21] HARBACH, M., HETTIG, M., WEBER, S., AND SMITH, M. Using personal examples to improve risk communication for security & privacy decisions. In *Proc. of the 32nd Annual ACM Conf. on Human Factors in Computing Systems* (New York, NY, USA, 2014), CHI '14, ACM, pp. 2647–2656.

[22] HARBACH, M., VON ZEZSCHWITZ, E., FICHTNER, A., DE LUCA, A., AND SMITH, M. It'sa hard lock life: A field study of smartphone (un) locking behavior and risk perception. In *Symposium on Usable Privacy and Security (SOUPS)* (2014).

[23] HASTIE, T., TIBSHIRANI, R., FRIEDMAN, J., AND FRANKLIN, J. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer 27*, 2 (2005), 83–85.

[24] HORNYACK, P., HAN, S., JUNG, J., SCHECHTER, S., AND WETHERALL, D. These aren't the droids you're looking for: retrofitting android to protect data from imperious applications. In *Proc. of the ACM Conf. on Comp. and Comm. Sec.* (New York, NY, USA, 2011), CCS '11, ACM, pp. 639–652.

[25] JUNG, J., HAN, S., AND WETHERALL, D. Short paper: Enhancing mobile application permissions with runtime feedback and constraints. In *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices* (New York, NY, USA, 2012), SPSM '12, ACM, pp. 45–50.

[26] KELLEY, P. G., CONSOLVO, S., CRANOR, L. F., JUNG, J., SADEH, N., AND WETHERALL, D. A conundrum of permissions: Installing applications on an android smartphone. In *Proc. of the 16th Intl. Conf. on Financial Cryptography and Data Sec.* (Berlin, Heidelberg, 2012), FC'12, Springer-Verlag, pp. 68–79.

[27] KELLEY, P. G., CRANOR, L. F., AND SADEH, N. Privacy as part of the app decision-making process. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2013), CHI '13, ACM, pp. 3393–3402.

[28] KLIEBER, W., FLYNN, L., BHOSALE, A., JIA, L., AND BAUER, L. Android taint flow analysis for app sets. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis* (New York, NY, USA, 2014), SOAP '14, ACM, pp. 1–6.

[29] LARSON, R., AND CSIKSZENTMIHALYI, M. New directions for naturalistic methods in the behavioral sciences. In *The Experience Sampling Method*, H. Reis, Ed. Jossey-Bass, San Francisco, 1983, pp. 41–56.

[30] LIN, J., SADEH, N., AMINI, S., LINDQVIST, J., HONG, J. I., AND ZHANG, J. Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing. In *Proc. of the 2012 ACM Conf. on Ubiquitous Computing* (New York, NY, USA, 2012), UbiComp '12, ACM, pp. 501–510.

[31] MALHOTRA, N. K., KIM, S. S., AND AGARWAL, J. Internet Users' Information Privacy Concerns (IUIPC): The Construct, The Scale, and A Causal Model. *Information Systems Research 15*, 4 (December 2004), 336–355.

[32] NISSENBAUM, H. Privacy as contextual integrity. *Washington Law Review 79* (February 2004), 119.

[33] NISSENBAUM, H. *Privacy in context: Technology, policy, and the integrity of social life*. Stanford University Press, 2009.

[34] O'GRADY, J. D. New privacy enhancements coming to ios 8 in the fall. `http://www.zdnet.com/new-privacy-enhancements-coming-to-ios-8-in-the-fall-7000030903/`, June 25 2014. Accessed: Nov. 11, 2014.

[35] PANDITA, R., XIAO, X., YANG, W., ENCK, W., AND XIE, T. WHYPER: Towards Automating Risk Assessment of Mobile Applications. In *Proc. of the 22nd USENIX Sec. Symp.* (Berkeley, CA, USA, 2013), SEC'13, USENIX Association, pp. 527–542.

[36] SARMA, B. P., LI, N., GATES, C., POTHARAJU, R., NITA-ROTARU, C., AND MOLLOY, I. Android permissions: A perspective combining risks and benefits. In *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies* (New York, NY, USA, 2012), SACMAT '12, ACM, pp. 13–22.

[37] SHEBARO, B., OLUWATIMI, O., MIDI, D., AND BERTINO, E. Identidroid: Android can finally wear its anonymous suit. *Trans. Data Privacy 7*, 1 (Apr. 2014), 27–50.

[38] SHKLOVSKI, I., MAINWARING, S. D., SKÚLADÓTTIR, H. H., AND BORGTHORSSON, H. Leakiness and creepiness in app space: Perceptions of privacy and mobile app use. In *Proc. of the 32nd Ann. ACM Conf. on Human Factors in Computing Systems* (New York, NY, USA, 2014), CHI '14, ACM, pp. 2347–2356.

[39] SPREITZENBARTH, M., FREILING, F., ECHTLER, F., SCHRECK, T., AND HOFFMANN, J. Mobile-sandbox: Having a deeper look into android applications. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing* (New York, NY, USA, 2013), SAC '13, ACM, pp. 1808–1815.

[40] STEVENS, R., GANZ, J., FILKOV, V., DEVANBU, P., AND CHEN, H. Asking for (and about) permissions used by android apps. In *Proc. of the 10th Working Conf. on Mining Soft- ware Repositories* (Piscataway, NJ, USA, 2013), MSR '13, IEEE Press, pp. 31–40.

[41] TAN, J., NGUYEN, K., THEODORIDES, M., NEGRON- ARROYO, H., THOMPSON, C., EGELMAN, S., AND WAGNER, D. The effect of developer-specified explanations for permission requests on smartphone user behavior. In *Proc. of the SIGCHI Conf. on Human Factors in Computing Systems* (2014).

[42] THOMPSON, C., JOHNSON, M., EGELMAN, S., WAGNER, D., AND KING, J. When it's better to ask forgiveness than get per- mission: Designing usable audit mechanisms for mobile permissions. In *Proceedings of the 2013 Symposium on Usable Privacy and Security (SOUPS)* (2013).

[43] VITTER, J. S. Random sampling with a reservoir. *ACM Trans. Math. Softw. 11*, 1 (Mar. 1985), 37–57.

[44] WEI, X., GOMEZ, L., NEAMTIU, I., AND FALOUTSOS, M. Permission evolution in the android ecosystem. In *Proceedings of the 28th Annual Computer Security Applications Conference* (New York, NY, USA, 2012), ACSAC '12, ACM, pp. 31–40.

[45] WOODRUFF, A., PIHUR, V., CONSOLVO, S., BRANDIMARTE, L., AND ACQUISTI, A. Would a privacy fundamentalist sell their dna for $1000...if nothing bad happened as a result? the westin categories, behavioral intentions, and consequences. In *Proceedings of the 2014 Symposium on Usable Privacy and Security* (2014), USENIX Association, pp. 1–18.

[46] XU, R., SAÏDI, H., AND ANDERSON, R. Aurasium: Practical policy enforcement for android applications. In *Proc. of the 21st USENIX Sec. Symp.* (Berkeley, CA, USA, 2012), Security'12, USENIX Association, pp. 27–27.

[47] ZHANG, Y., YANG, M., XU, B., YANG, Z., GU, G., NING, P., WANG, X. S., AND ZANG, B. Vetting undesirable behaviors in android apps with permission use analysis. In *Proc. of the ACM Conf. on Comp. and Comm. Sec.* (New York, NY, USA, 2013), CCS '13, ACM, pp. 611–622.

[48] ZHU, H., XIONG, H., GE, Y., AND CHEN, E. Mobile app recommendations with security and privacy awareness. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2014), KDD '14, ACM, pp. 951–960.

A Invisible requests

Following list shows the set of applications that have requested the most number of permissions while executing invisibly to the user and the most requested permission types by each respective application.

- *Facebook App*— ACCESS NETWORK STATE, ACCESS FINE LOCATION, ACCESS WIFI STATE ,WAKE LOCK,
- *Google Location*—WAKE LOCK, ACCESS FINE LOCATION, GET ACCOUNTS, ACCESS COARSE LOCATION,
- *Facebook Messenger*—ACCESS NETWORK STATE, ACCESS WIFI STATE, WAKE LOCK, READ PHONE STATE,
- *Taptu DJ*—ACCESS NETWORK STATE, INTERNET, NFC
- *Google Maps*—ACCESS NETWORK STATE, GET ACCOUNTS, WAKE LOCK, ACCESS FINE LOCATION,
- *Google (Gapps)*—WAKE LOCK, ACCESS FINE LOCATION, AUTHENTICATE ACCOUNTS, ACCESS NETWORK STATE,
- *Fouraquare*—ACCESS WIFI STATE, WAKE LOCK, ACCESS FINE LOCATION, INTERNET,
- *Yahoo Weather*—ACCESS FINE LOCATION, ACCESS NETWORK STATE, INTERNET, ACCESS WIFI STATE,

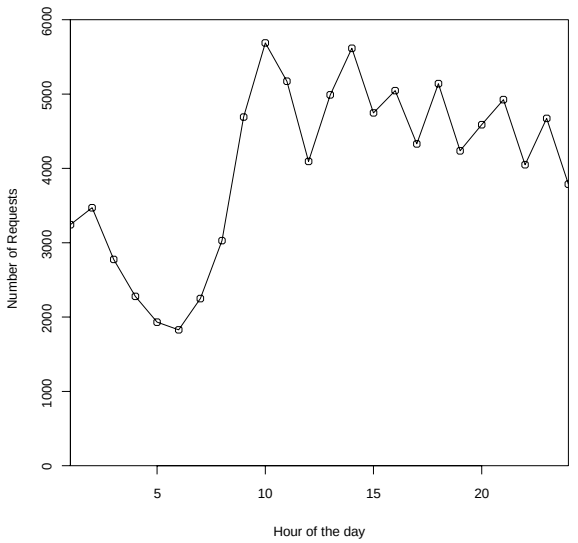
- *Devexpert Weather*—ACCESS NETWORK STATE, INTERNET, ACCESS FINE LOCATION,
- *Tile Game(Umoni)*—ACCESS NETWORK STATE, WAKE LOCK, INTERNET, ACCESS WIFI STATE,

Following is the most frequently requested permission type by applications while running invisibly to the user and the applications who requested the respective permission type most.

- *ACCESS_NETWORK_STATE*— Facebook App, Google Maps, Facebook Messenger, Google (Gapps), Taptu - DJ
- *WAKE_LOCK*—Google (Location), Google (Gapps), Google (GMS), Facebook App, GTalk.
- *ACCESS_FINE_LOCATION*—Google (Location), Google (Gapps), Facebook App, Yahoo Weather, Rhapsody (Music)
- *GET_ACCOUNTS*—Google (Location), Google (Gapps), Google (Login), Google (GM), Google (Vending)
- *ACCESS_WIFI_STATE*—Google (Location), Google (Gapps), Facebook App, Foursqaure, Facebook Messenger
- *UPDATE_DEVICE_STATS*—Google (SystemUI), Google (Location), Google (Gapps)
- *ACCESS_COARSE_LOCATION*—Google (Location), Google (Gapps), Google (News), Facebook App, Google Maps
- *AUTHENTICATE_ACCOUNTS*—Google (Gapps), Google (Login), Twitter, Yahoo Mail, Google (GMS)
- *READ_SYNC_SETTINGS*—Google (GM), Google (GMS), android.process.acore, Google (Email), Google (Gapps)
- *INTERNET*—Google (Vending), Google (Gapps), Google (GM), Facebook App, Google (Location)

B Distribution of Requests

The following graph shows the distribution of requests throughout a given day averaged across the data set.



C Permission Type Breakdown

This table lists the most frequently used permissions during the study period. (per user / per day)

Permission Type	Requests
ACCESS_NETWORK_STATE	41077
WAKE_LOCK	27030
ACCESS_FINE LOCATION	7400
GET_ACCOUNTS	4387
UPDATE_DEVICE_STATS	2873
ACCESS_WIFI STATE	2092
ACCESS_COARSE LOCATION	1468
AUTHENTICATE_ACCOUNTS	1335
READ_SYNC SETTINGS	836
VIBRATE	740
INTERNET	739
READ_SMS	611
READ_PHONE STATE	345
STATUS BAR	290
WRITE_SYNC SETTINGS	206
CHANGE_COMPONENT_ENABLED_STATE	197
CHANGE_WIFI STATE	168
READ_CALENDAR	166
ACCOUNT_MANAGER	134
ACCESS_ALL_DOWNLOADS	127
READ_EXTERNAL_STORAGE	126
USE_CREDENTIALS	101
READ_LOGS	94

D User Application Breakdown

This table shows the applications that most frequently requested access to protected resources during the study period. (per user / per day)

Application Name	Requests
facebook.katana	40041
google.process.location	32426
facebook.orca	24702
taptu.streams	15188
google.android.apps.maps	6501
google.process.gapps	5340
yahoo.mobile.client.android.weather	5505
tumblr	4251
king.farmheroessaga	3862
joelapenna.foursquared	3729
telenav.app.android.scout us	3335
devexpert.weather	2909
ch.bitspin.timely	2549
umonistudio.tile	2478
king.candycrushsaga	2448
android.systemui	2376
bambuna.podcastaddict	2087
contapps.android	1662
handcent.nextsms	1543
foursquare.robin	1408
qisiemoji.inputmethod	1384
devian.tubemate.home	1296
lookout	1158

Appendix B The Feasbility of Dynamically Granted Permissions: Aligning Mobile Privacy with User Preferences

The Feasibility of Dynamically Granted Permissions: Aligning Mobile Privacy with User Preferences

Primal Wijesekera[1], Arjun Baokar[2], Lynn Tsai[2], Joel Reardon[2],
Serge Egelman[2], David Wagner[2], and Konstantin Beznosov[1]
[1]University of British Columbia, Vancouver, Canada,
{primal,beznosov}@ece.ubc.ca
[2]University of California, Berkeley, Berkeley, USA,
{arjunbaokar,lynntsai,joel.reardon}@berkeley.edu, {egelman,daw}@cs.berkeley.edu

***Abstract*—Current smartphone operating systems regulate application permissions by prompting users on an ask-on-first-use basis. Prior research has shown that this method is ineffective because it fails to account for context: the circumstances under which an application first requests access to data may be vastly different than the circumstances under which it subsequently requests access. We performed a longitudinal 131-person field study to analyze the contextuality behind user privacy decisions to regulate access to sensitive resources. We built a classifier to make privacy decisions on the user's behalf by detecting when context has changed and, when necessary, inferring privacy preferences based on the user's past decisions and behavior. Our goal is to automatically grant appropriate resource requests without further user intervention, deny inappropriate requests, and only prompt the user when the system is uncertain of the user's preferences. We show that our approach can accurately predict users' privacy decisions 96.8% of the time, which is a four-fold reduction in error rate compared to current systems.**

I. Introduction

One of the roles of a mobile application platform is to help users avoid unexpected or unwanted use of their personal data [12]. Mobile platforms currently use permission systems to regulate access to sensitive resources, relying on user prompts to determine whether a third-party application should be granted or denied access to data and resources. One critical caveat in this approach, however, is that mobile platforms seek the consent of the user the first time a given application attempts to access a certain data type and then enforce the user's decision for all subsequent cases, regardless of the circumstances surrounding each access. For example, a user may grant an application access to location data because she is using location-based features, but by doing this, the application can subsequently access location data for behavioral advertising, which may violate the user's preferences.

Earlier versions of Android (5.1 and below) asked users to make privacy decisions during application installation as an all-or-nothing ultimatum (ask-on-install): either all requested permissions are approved or the application is not installed. Previous research showed that few people read the requested permissions at install-time and even fewer correctly understood them [17]. Furthermore, install-time permissions do not present users with the context in which those permission will be exercised, which may cause users to make suboptimal decisions not aligned with their actual preferences. For example, Egelman et al. observed that when an application requests access to location data without providing context, users are just as likely to see this as a signal for desirable location-based features as they are an invasion of privacy [11]. Asking users to make permission decisions at runtime—at the moment when the permission will actually be used by the application—provides more context (i.e., what they were doing at the time that data was requested) [15]. However, due to the high frequency of permission requests, it is not feasible to prompt the user every time data is accessed [43].

In iOS and Android M, the user is now prompted at runtime the first time an application attempts to access one of a set of "dangerous" permission types (e.g., location, contacts, etc.). This *ask-on-first-use* (AOFU) model is an improvement over ask-on-install (AOI). Prompting users the first time an application uses one of the designated permissions gives users a better sense of context: their knowledge of what they were doing when the application first tried to access the data should help them determine whether the request is appropriate. Despite that, Wijesekera et al. showed that AOFU fails to meet user expectations over half the time. This is because AOFU does not account for the varying contexts of future requests [43].

The notion of *contextual integrity* suggests that many permission models fail to protect user privacy because they fail to account for the context surrounding data flows [34]. That is, privacy violations occur when sensitive resources are used in ways that defy users' expectations. We posit that more effective permission models must focus on whether resource accesses are likely to defy users' expectations in a given context—not simply whether the application was authorized to receive data the first time it asked for it. Thus, the challenge for system designers is to correctly infer when the context surrounding a data request has changed, and whether the new context is likely to be deemed "appropriate" or "inappropriate" for the given user. Dynamically regulating data access based on the context requires more user involvement to understand users' contextual preferences. If users are asked to make privacy decisions too frequently, or under circumstances that are seen as low-risk, they may become habituated to future, more serious, privacy

decisions. On the other hand, if users are asked to make too few privacy decisions, they may find that the system has acted against their wishes. Thus, our goal is to automatically determine *when* and under *what* circumstances the system presents users with runtime prompts.

To this end, we collected real-world Android usage data in order to explore whether we could infer users' future privacy decisions based on their past privacy decisions, contextual circumstances surrounding applications' data requests, and users' behavioral traits. We conducted a field study where 131 participants used Android phones that were instrumented to gather data over an average of 32 days per participant. Also, their phones periodically prompted them to make privacy decisions when applications used sensitive permissions, and we logged their decisions. Overall, participants wanted to block 60% of these requests. We found that AOFU yields 84% accuracy, i.e., its policy agrees with participants' prompted responses 84% of the time. AOI achieves only 25% accuracy. We designed new techniques that use machine learning to automatically predict how users would respond to prompts, so that we can avoid prompting them in most cases, thereby reducing user burden. Our classifier uses the user's past decisions in similar situations to predict their response to a particular permission request. The classifier outputs a prediction and a confidence score; if the classifier is sufficiently confident, we use its prediction, otherwise we prompt the user for their decision. We also incorporate information about the user's behavior in other security and privacy situations to make inferences about their preferences: whether they have a screen lock activated, how often they visit HTTPS websites, and so on. We show that our scheme achieves 96.8% accuracy (a 4 reduction in error rate over AOFU) with significantly less user involvement than the *status quo*.

The specific contributions of our work are the following:

- We conducted the first known large-scale study on quantifying the effectiveness of ask-on-first-use permissions.
- We show that a significant portion of the studied participants make contextual decisions on permissions— the foreground application and the visibility of the permission-requesting application are strong cues participants used to make contextual decisions.
- We show how a machine-learned model can incorporate context and better predict users' privacy decisions.
- To our knowledge, we are the first to use passively observed traits to infer future privacy decisions on a case-by-case basis at runtime.

II. Related Work

There is a large body of work demonstrating that install- time prompts fail because users do not understand or pay attention to them [19], [23], [42]. When using install-time prompts, users often do not understand which permission types correspond to which sensitive resources and are surprised by the ability of background applications to collect information [17], [22], [41]. Applications also transmit a large amount of location or other sensitive data to third parties without user consent [12]. When possible risks associated with these requests are revealed to users, their concerns range from being annoyed to wanting to seek retribution [16].

To mitigate some of these problems, systems have been developed to track information flows across the Android system [12], [18], [24] or introduce finer-grained permission control into Android [2], [21], [39], but many of these solutions increase user involvement significantly, which can lead to habituation. Additionally, many of these proposals are useful only to the most-motivated or technically savvy users. For example, many such systems require users to configure complicated control panels, which many are unlikely to do [45]. Other approaches involve static analysis in order to better understand how applications *could* request information [4], [8], [14], but these say little about how applications *actually* use information. Dynamic analysis improves upon this by allowing users to see how often this information is requested in real time [12], [40], [43], but substantial work is likely needed to present that information to average users in a meaningful way. Solutions that require user interruptions need to also minimize user intervention in order to prevent habituation.

Other researchers have developed recommendation systems to recommend applications based on users' privacy preferences [46], or detect privacy violations and suggest preferences based on crowdsourcing [1], [27], but such approaches often do not take individual user differences into account without significant user intervention. Systems have also been developed to predict what users would share on mobile social networks [7], which suggests that future systems could potentially infer what information users would be willing to share with third-party applications. By requiring users to self-report privacy preferences, clustering algorithms have been used to define user privacy profiles even in the face of diverse preferences [26], [38]. However, researchers have found that the order in which information is requested has an impact on prediction accuracy [44], which could mean that such systems are only likely to be accurate when they examine actual user behavior over time (as opposed to one-time self-reports).

Liu et al. clustered users by privacy preferences and used ML techniques to predict whether to allow or deny an application's request for sensitive user data [29]. Their dataset, however, was collected from a set of highly privacy-conscious individuals: those who choose to install a permission-control mechanism. Furthermore, the researchers removed "conflicting" user decisions, in which a user chose to deny a permission for an application, and then later chose to allow it. These conflicting decisions, however, do not represent noisy data. They occur nearly 50% of the time in the real world [43], and accurately reflect the nuances of user privacy preferences. Models must therefore account for them. In fact, previous work found that users commonly reassess privacy preferences after usage [3]. Liu et al. also expect users to make 10% of permission decisions manually, which, based on field study results from Wijesekera et al., would result in being prompted every three minutes [43]. This is obviously impractical. Our goal is to design a system that can automatically make decisions on

behalf of users, that accurately models their preferences, while also not over-burdening them with repeated requests.

Closely related to this work, Liu et al. [28] performed a field study to measure the effectiveness of a Privacy Assistant that offers recommendations to users on privacy settings that they could adopt based on each user's privacy profile—the privacy assistant predicts what the user might want based on the inferred privacy profile and static analysis of the third-party application. While this approach increased user awareness on resource usage, the recommendations are static: they do not consider each application's access to sensitive data on a case-by-case basis. Such a coarse-grained approach goes against previous work suggesting that people do want to vary their decisions based on contextual circumstances [43]. A blanket approval or denial of a permission to a given application carries a considerable risk of privacy violations or loss of desired functionality. In contrast, our work uses dynamic analysis to infer the appropriateness of each given request by considering the surrounding contextual cues and how the user has behaved in similar situations in the past. As with Liu et al., their dataset was also collected from privacy-conscious and considerably tech-savvy individuals, which may limit the generalization of their results. The field study we conduct in our work uses a more representative sample.

Nissenbaum's theory of contextual integrity suggests that permission models should focus on information flows that are likely to defy user expectations [34]. There are three main components involved in deciding the appropriateness of a flow [6]: the context in which the resource request is made, the role played by the requesting application under the current context, and the type of resource being accessed. Neither previous nor currently deployed permission models take all three factors into account. This model could be used to improve permission models by automatically granting access to data when the system determines that it is appropriate, denying access when it is inappropriate, and prompting the user only when a decision cannot be made automatically, thereby reducing user burden.

Access Control Gadgets (ACGs) were proposed as a mechanism to tie sensitive resource access to certain UI elements [32], [35]–[37]. Authors posit that such an approach will increase user expectations, as a significant portion of participants expected a UI interaction before a sensitive resource usage, giving users an implicit mechanism to control access and increasing awareness on resource usage. The biggest caveat in this approach is that tying a UI interaction to each sensitive resource access is impossible in practice because resources are accessed at a high frequency [43], and because many legitimate resource accesses occur without user initiation [15]. Wijesekera et al. performed a field study [43] to operationalize the notion of "context," to allow an operating system to differentiate between appropriate and inappropriate data requests by a single application for a single data type. They found that users' decisions to allow a permission request significantly correlated with that application's visibility. They posit that this visibility is a strong contextual cue that influences users' responses to permission prompts. They also observed that privacy decisions were highly nuanced, demonstrating that a one-size-fits-all model is unlikely to be sufficient; a given information flow may be deemed appropriate by one user but not by another user. They recommended applying machine learning in order to infer individual users' privacy preferences. To achieve this, research is needed to determine what factors affect user privacy decisions and how to use those factors to make privacy decisions on the user's behalf. While we cannot automatically capture everything involved in Nissenbaum's notion of *context*, we can try to detect when context has likely changed (insofar as to decide whether a different privacy decision should be made for the same application and data type), by seeing whether the circumstances surrounding a data request are similar to previous requests.

Permission Type	Activity
ACCESS_WIFI_STATE	View nearby SSIDs
NFC	Communicate via NFC
READ_HISTORY_BOOKMARKS	Read users' browser history
ACCESS_FINE_LOCATION	Read GPS location
ACCESS_COARSE_LOCATION	Read network-inferred location (i.e., cell tower and/or WiFi)
LOCATION_HARDWARE	Directly access GPS data
READ_CALL_LOG	Read call history
ADD_VOICEMAIL	Read call history
READ_SMS	Read sent/received/draft SMS
SEND_SMS	Send SMS
*INTERNET	Access Internet when roaming
*WRITE_SYNC_SETTINGS	Change application sync settings when roaming

TABLE I
FELT ET AL. PROPOSED GRANTING A SELECT SET OF 12 PERMISSIONS AT RUNTIME SO THAT USERS HAVE CONTEXTUAL INFORMATION TO INFER WHY THE DATA MIGHT BE NEEDED [15]. OUR INSTRUMENTATION OMITS THE LAST TWO PERMISSION TYPES (INTERNET & WRITE_SYNC_SETTINGS) AND RECORDS INFORMATION ABOUT THE OTHER 10.

III. METHODOLOGY

We collected data from 131 participants to understand what factors could be used to infer whether a permission request is likely to be deemed appropriate by the user.

Previous work by Felt et al. made the argument that certain permissions are appropriate for runtime prompts, because they protect sensitive resources and because viewing the prompt at runtime imparts additional contextual information about why an application might need the permission [15]. Similarly, Thompson et al. showed that other permission requests could be replaced with audit mechanisms, because they represent either reversible changes or are sufficiently low risk to not warrant habituating the user to prompts [41]. We collected information about 10 of the 12 permissions Felt et al. suggest are best-suited for runtime prompts. We omitted INTERNET and WRITE_SYNC_SETTINGS, because those permissions only warrant runtime prompts if the user is roaming and we did not expect any participant to be roaming during the study period, and focused on the remaining 10 permission types (Table I). While there are many other sensitive permissions beyond this

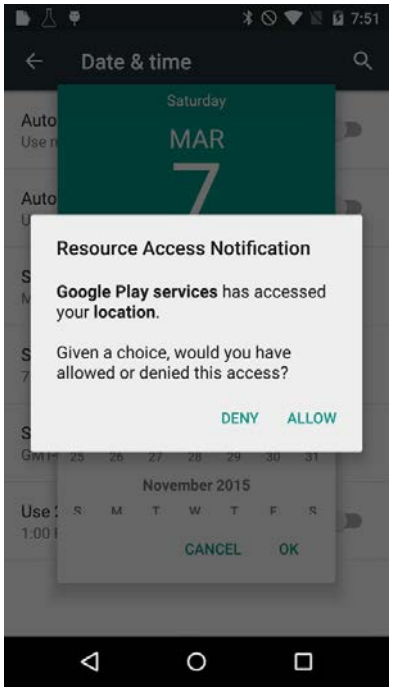


Fig. 1. A screenshot of an ESM prompt.

Type	Event Recorded
Behavioral Instrumentation	Changing developer options
	Opening/Closing security settings
	Changing security settings
	Enabling/Disabling NFC
	Changing location mode
	Opening/Closing location settings
	Changing screen-lock type
	Use of two factor authentication
	Log initial settings information
	User locks the screen
	Screen times out
	App locks the screen
	Audio mode changed
	Enabling/Disabling speakerphone
	Connecting/Disconnecting headphones
	Muting the phone
	Taking an audio call
	Taking a picture (front- vs. rear-facing)
	Visiting an HTTPS link in Chrome
	Responding to a notification
	Unlocking the phone
Runtime Information	An application changing the visibility
	Platform switches to a new activity
Permission Requests	An app requests a sensitive permission
	ESM prompt for a selected permission

TABLE II
INSTRUMENTED EVENTS THAT FORM OUR FEATURE SET

set, Felt et al. concluded that the others are best handled by other mechanisms (e.g., install-time prompts, ACGs, etc.).

We used the Experience Sampling Method (ESM) to collect ground truth data about users' privacy preferences [20]. ESM involves repeatedly questioning participants *in situ* about a recently observed event; in this case, we probabilistically asked them about an application's recent access to data on their phone, and whether they would have permitted it if given the choice. We treated participants' responses to these ESM probes as our main dependent variable (Figure 1).

We also instrumented participants' smartphones to obtain data about their privacy-related behaviors and the frequency with which applications accessed protected resources. The instrumentation required a set of modifications to the Android operating system and flashing a custom Android version onto participants' devices. To facilitate such experiments, the University of Buffalo offers non-affiliated academic researchers access to the PhoneLab panel [33], which consists of more than 200 participants. All of these participants had LG Nexus 5 phones running Android 5.1.1 and the phones were periodically updated over-the-air (OTA) with custom modifications to the Android operating system. Participants can decide when to install the OTA update, which marks their entry into new experiments. During our experiment period, different participants installed the OTA update with our instrumentation at different times, thus we have neither data on all PhoneLab participants nor data for the entire period. Our OTA update was available to participants for a period of six weeks, between February 2016 and March 2016. At the end of the study period, we emailed participants a link to an exit survey to collect demographic information. Our study received institutional review board (IRB) approval.[1]

A. Instrumentation

The goal of our instrumentation was to collect as much runtime and behavioral data as could be observed from the Android platform, with minimal performance cost. We collected three categories of data: behavioral information, runtime information, and user decisions. We made no modifications to any third-party application code; our dynamic analysis techniques could be used on any third-party Android application.

Table II contains the complete list of behavioral and runtime events our instrumentation recorded. The behavioral data fell under several categories, all chosen based on several hypotheses that we had about the types of behaviors that might correlate with privacy preferences: web-browsing habits, screen locking behavior, third-party application usage behavior, audio preferences, call habits, camera usage patterns, and behavior related to security settings. For example, we hypothesized that someone who manually locks their device screen are more privacy-conscious than someone who lets it time out.

We also collected runtime information about the context of each permission request, including the visibility of the requesting application at the time of request, what the user was doing when the request was made (i.e., the name of the foreground application), and the exact Android API function invoked by the application to determine what information was requested. The visibility of an application reflects the extent to which the

[1]Approved by the UC Berkeley IRB under protocol #2013-02-4992

user was likely aware that the application was running; if the application was in the foreground, the user had cues that the application was running, but if it was in the background, then the user was likely not aware that the application was running and therefore might find the permission request unexpected—some background services can still be visible to the user due to on-screen notification or other cues that could be perceptible. We monitored processes' memory priority levels to determine the visibility of all Android processes. We also collected information about which Android `Activity` was active in the application.[2]

Once per day we *probabilistically* selected one of these permission requests and prompted the user about them at runtime (Figure 1). We used weighted reservoir sampling to select a permission request to prompt about. We weight the combination of *application, permission, visibility* based on their frequency of occurrence seen by the instrumentation; the most-frequent combination has a higher probability of being shown to participants using ESM. We prompted participants a maximum of three times for each unique combination. We tuned the wording of the prompt to make it clear that the request had just occurred and their response would not affect the system (a deny response would not actually deny data). These responses serve as the ground truth for all the analysis mentioned in the remainder of the paper.

The intuition behind using weighted reservoir sampling is to focus more on the frequently occurring permission requests over rare ones. Common permission requests contribute most to user habituation due to their high frequency. Thus, it is more important to learn about user privacy decisions on highly frequent permission requests over the rare ones, which might not risk user habituation or annoyance (and the context of rare requests may be less likely to change).

B. Exit Survey

At the end of our data collection period, PhoneLab staff emailed participants a link to our online exit survey, which they were incentivized to complete with a raffle for two $100 Amazon gift cards. The survey gathered demographic information and qualitative information on their privacy preferences. Of the 203 participants in our experiment, 53 fully completed the survey, and another 14 partially completed it. Of the 53 participants to fully complete the survey, 21 were male, 31 were female, and 1 undisclosed. Participants ranged from 20 to 72 years of age (μ = 40.83, CJ = 14.32). Participants identified themselves as 39.3% staff, 32.1% students, 19.6% faculty, and 9% other. Only 21% of the survey respondents had an academic qualification in STEM, which suggests that the sample is unlikely to be biased towards tech-savvy users.

C. Summary

We collected data from February 5 to March 17, 2016. PhoneLab allows any participant to opt-out of an experiment at any time. Thus, of the 203 participants who installed our custom Android build, there were 131 who used it for more than 20 days. During the study period, we collected 176M events across all participants (31K events per participant/day). Our dataset consists of 1,686 unique applications and 13K unique activities. Participants also responded to 4,636 prompts during the study period. We logged 96M sensitive permission requests, which translates to roughly one sensitive permission request every 6 seconds per participant. For the remainder of the paper, we only consider the data from the 131 participants who used the system for at least 20 days, which corresponds to 4,224 ESM prompts.

Of the 4,224 prompts, 55.3% were in response to ACCESS_WIFI_STATE, when trying to access WiFi SSID information that could be used to infer the location of the smartphone; 21.0%, 17.3%, 5.08%, 0.78%, and 0.54% were from accessing location directly, reading SMS, sending SMS, reading call logs, and accessing browser history, respectively. A total of 137 unique applications triggered prompts during the study period. Of the 4,224 prompts, participants wanted to deny 60.01% of them, and 57.65% of the prompts were shown when the requesting application was running in the foreground or the user had visual cues that the application was running (e.g., notifications). A Wilcoxon signed rank test with continuity correction revealed a statistically significant difference in participants' desire to allow or deny a permission request based on the visibility of the requesting application ($p < 0.0152$, $r = 0.221$), which corroborates previous findings [43].

IV. Types of Users

We hypothesized that there may be different types of users based on how they want to disclose their private information to third parties. It is imperative to identify these different sub-populations since different permission models affect users differently based on their privacy preferences; performance numbers averaged across a user population could be misleading since different sub-populations might react differently to the same permission model.

While our study size was too small to effectively apply clustering techniques to generate classes of users, we did find a meaningful distinction using the denial rate (i.e., the percentage of prompts to which users wanted to deny access). We aggregated users by their denial rate in 10% increments and examined how these different participants considered the surrounding contextual circumstances in their decisions.

We discovered that application visibility was a significant factor for users with a denial rate of 10–90%, but not for users with a denial rate of 0–10% or 90–100%. We call the former group *Contextuals*, as they seem to care about the surrounding context (i.e., they make nuanced decisions, allowing or denying a permission request based on whether they had contextual cues that indicated that the requesting application was running), and the latter group *Defaulters*, because they seem to simply always allow or always deny requests, regardless of contextual cues.

Defaulters accounted for 53% of 131 participants and *Contextuals* accounted for 47%. A Wilcoxon signed-rank test with

[2]An Android `Activity` represents the application screen and UI elements currently exposed to the user.

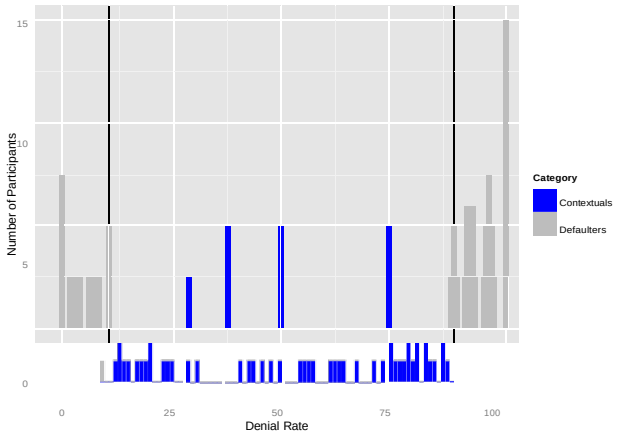


Fig. 2. Histogram of users based on their denial rate. *Defaulters* tended to allow or deny almost all requests without regard for contextual cues, whereas *Contextuals* considered the visibility of the requesting application.

Policy	Contextuals	Defaulters	Overall	Prompts
AOI	44.11%	6.00%	25.00%	0.00
AOFU-AP	**64.49%**	**93.33%**	**84.61%**	**12.34**
AOFU-APV	64.28%	92.85%	83.33%	15.79
AOFU-A_FPV	66.67%	98.95%	84.61%	16.91
AOFU-VP	58.65%	94.44%	78.04%	6.43
AOFU-VA	63.39%	93.75%	84.21%	12.24
AOFU-A	64.27%	93.54%	83.33%	9.06
AOFU-P	57.95%	95.45%	82.14%	3.84
AOFU-V	52.27%	95.34%	81.48%	2.00

TABLE III

THE ACCURACY AND NUMBER OF DIFFERENT POSSIBLE ASK-ON-FIRST- USE COMBINATIONS. A: APPLICATION REQUESTING THE PERMISSION, P: PERMISSION TYPE REQUESTED, V: VISIBILITY OF THE APPLICATION REQUESTING THE PERMISSION, A_F: APPLICATION RUNNING IN THE FOREGROUND WHEN THE REQUEST IS MADE. AOFU-AP IS THE POLICY USED IN ANDROID MARSHMALLOW I.E., ASKING (PROMPTING) THE USER FOR EACH UNIQUE APPLICATION, PERMISSION COMBINATION. THE TABLE ALSO DIFFERENTIATES POLICY NUMBERS BASED ON THE SUBPOPULATION OF *Contextuals*, *Defaulters*, AND ACROSS ALL USERS.

continuity correction revealed a statistically significant difference in *Contextuals'* responses based on requesting application visibility ($p < 0.013$, $r = 0.312$), while for *Defaulters* there was no statistically significant difference ($p = 0.227$). That is, *Contextuals* used visibility as a contextual cue, when deciding the appropriateness of a given permission request, whereas *Defaulters* did not vary their decisions based on this cue. Figure 2 shows the distribution of users based on their denial rate. Vertical lines indicate the borders between *Contextuals* and *Defaulters*.

In the remainder of the paper, we use our *Contextuals*–*Defaulters* categorization to measure how current and proposed models affect these two sub-populations, issues unique to these sub-populations, and ways to address these issues.

V. ASK-ON-FIRST-USE PERMISSIONS

Ask-on-first-use (AOFU) is the current Android permission model, which was first adopted in Android 6.0 (Marshmallow). AOFU prompts the user whenever an application requests a *dangerous* permission for the first time [9]; the user's response to this prompt is thereafter applied whenever the same application requests the same permission. As of March 2017, only 34.1% of Android users have Android Marshmallow or a higher version [10], and among these Marshmallow users, those who upgraded from a previous version only see runtime permission prompts for freshly-installed applications.

For the remaining 65.9% of users, the system policy is ask-on-install (AOI), which automatically allows all runtime permission requests. During the study period, all of our participants had AOI running as the default permission model. Because all runtime permission requests are allowed in AOI, any of our ESM prompts that the user wanted to deny correspond to mispredictions under the AOI model (i.e., the AOI model granted access to the data against users' actual preferences).

Table III shows the expected median accuracy for AOI, as well as several other possible variants that we discuss in this section. The low median accuracy for *Defaulters* was due to the significant number of people who simply denied most of the prompts. The prompt count is zero for AOI because it does not prompt the user during runtime; users are only shown permission prompts at installation.

More users will have AOFU in the future, as they upgrade to Android 6.0 and beyond. To the best of our knowledge, no prior work has looked into quantifying the effectiveness of AOFU systematically; this section presents analysis of AOFU based on prompt responses collected from participants and creates a baseline against which to measure our system's improvement. We simulate how AOFU performs through our ESM prompt responses. Because AOFU is deterministic, each user's response to the first prompt for each *application:permission* combination tells us how the AOFU model would respond for subsequent requests by that same combination. For participants who responded to more than one prompt for each combination, we can quantify how often AOFU would have been correct for subsequent requests. Similarly, we also measure the accuracy for other possible policies that the platform could use to decide whether to prompt the user. For example, the status quo is for the platform to prompt the user for each new *application:permission* combination, but how would accuracy (and the number of prompts shown) change if the policy were to prompt on all new combinations of *application:permission:visibility*? Table III shows the expected median accuracy[3] for each policy based on participants' responses. For each policy, *A* represents the application requesting the permission, *P* represents the requested permission, *V* represents the visibility of the requesting application, and A_F represents the application running in the foreground when a sensitive permission request was made. For instance, AOFU-AP is the policy where the user will be prompted for each new instance of an *application:permission* combination, which the Android 6.0 model employs. The last column shows the number of runtime prompts a participant would see under each policy over the duration of the study, if that policy were to be

[3]The presented numbers—except for average prompt count, which was normally distributed—are median values, because the distributions were skewed.

implemented. Both AOFU-AP and AOFU-A$_F$PV show about a 4.9–reduction in error rate compared to AOI; AOFU-A$_F$PV would require more prompts over AOFU-AP, though yields a similar overall accuracy rate. [4] Moving forward, we focus our analysis only on AOFU-AP (i.e., the current standard).

Instances where the user wants to deny a permission and the policy instead allows it (false positives) are *privacy violations*, because they expose more information to the application than the user desires. Instances where the user wants to allow a permission, but the policy denies it (false negatives) are *functionality losses*. This is because the application is likely to lose some functionality that the user desired when it is incorrectly denied a permission. Privacy violations and functionality losses were approximately evenly split between the two categories for AOFU-AP: median privacy violations and median functionality losses were 6.6% and 5.0%, respectively. The AOFU policy works well for *Defaulters* because, by definition, they tend to be consistent after their initial responses for each combination. In contrast, the decisions of *Contextuals* vary due to other factors beyond just the requesting application and the requested permission type. Hence, the accuracy of AOFU for *Contextuals* is significantly lower than the accuracy for *Defaulters*. This distinction shows that learning privacy preferences for a significant portion of users requires a deeper understanding of factors affecting their decisions, such as behavioral tendencies and contextual cues. As Table III suggests, superficially adding more contextual variables (such as visibility of the requesting application) does not necessarily help to increase the accuracy of the AOFU policy.

The context in which users are prompted under AOFU might be a factor affecting its ability to predict subsequent instances. In previous work [43], we found that the visibility of the requesting application is a strong contextual cue users use to vary their decisions. During the study period, under the AOFU-AP policy, 60% of the prompts could have occurred when the requesting application was visible to the participant—these prompts had an accuracy of 83.3% in predicting subsequent instances. In instances where participants were prompted when the requesting application was running invisibly to the user, AOFU-AP had an accuracy of 93.7% in predicting subsequent instances. A Wilcoxon signed-ranks test, however, did not reveal a statistically significant difference ($p < 0.3735$).

Our estimated accuracy numbers for AOFU may be inflated because AOFU in deployment (Android 6 and above) does not filter permission requests that do not reveal any sensitive information. For example, an application can request the ACCESS_FINE_LOCATION permission to check whether the phone has a specific location provider, which does not leak sensitive information. Our AOFU simulation uses the invoked function to determine if sensitive data was *actually* accessed, and only prompts in those cases (in the interest of avoiding any false positives), a distinction that AOFU in Android does not make. Thus, an Android user would see a permission request prompt when the application examines the list of location providers, and if the permission is granted, would not subsequently see prompts when location data is actually captured. Previous work found that 79% of first-time permission requests do not reveal any sensitive information [43], and nearly 33.9% of applications that request these sensitive permission types do not access sensitive data at all. The majority of AOFU prompts in Marshmallow are therefore effectively false positives, which incorrectly serve as the basis for future decisions. Given this, AOFU's average accuracy is likely less than the numbers presented in Table III. We therefore consider our estimates of AOFU to be an upper bound.

[4] While AOFU-A$_F$PV has greater *median* accuracy when examining *Defaulters* and *Contextuals* separately, because the distributions are skewed, the median overall accuracy is identical to AOFU-AP when combining the groups.

VI. Learning Privacy Preferences

Table III shows that a significant portion of users (the 47% classified as *Contextuals*) make privacy decisions that depend on factors other than the application requesting the permission, the permission requested, and the visibility of the requesting application. To make decisions on behalf of the user, we must understand what other factors affect their privacy decisions. We built a machine learning model trained and tested on our labeled dataset of 4,224 prompts collected from 131 users over the period of 42 days. This approach is equivalent to training a model based on runtime prompts from hundreds of users and using it to predict those users' future decisions.

We focus the scope of this work by making the following assumptions. We assume that the platform, i.e., the Android OS, is trusted to manage and enforce permissions for applications. We assume that applications must go through the platform's permission system to gain access to protected resources. We assume that we are in a non-adversarial machine-learning setting wherein the adversary does not attempt to circumvent the machine-learned classifier by exploiting knowledge of its decision-making process—though we do present a discussion of this problem and potential solutions in Section IX.

A. Feature Selection

Using the behavioral, contextual, and aggregate features shown in Table II, we constructed 16K candidate features, formed by combinations of specific applications and actions. We then selected 20 features by measuring Gini importance through random forests [30], significance testing for correlations, and singular value decomposition (SVD). SVD was particularly helpful to address the sparsity and high dimensionality issues caused by features generated based on application and activity usage. Table IV lists the 20 features used in the rest of this work.

The behavioral features (B) that proved predictive relate to browsing habits, audio/call traits, and locking behavior. All behavioral features were normalized per day/user and were scaled in the actual model. Features relating to browsing habits included the number of websites visited, the proportion of HTTPS-secured links visited, the number of downloads, and proportion of sites visited that requested location access. Features relating to locking behavior included whether users employed a passcode/PIN/pattern, the frequency of screen

Feature Group	Feature	Type
Behavioral Features (B)	Number of times a website is loaded to the Chrome browser.	Numerical
	Out of all visited websites, the proportion of HTTPS-secured websites.	Numerical
	The number of downloads through Chrome.	Numerical
	Proportion of websites requested location through Chrome.	Numerical
	Number of times PIN/Password was used to unlock the screen.	Numerical
	Amount of time spent unlocking the screen.	Numerical
	Proportion of times screen was timed out instead of pressing the lock button.	Numerical
	Frequency of audio calls.	Numerical
	Amount of time spent on audio calls.	Numerical
	Proportion of time spent on silent mode.	Numerical
Runtime Features (R1)	Application visibility (True/False)	Categorical
	Permission type	Categorical
	User ID	Categorical
	Time of day of permission request	Numerical
Aggregated Features (A)	Average denial rate for (A1) application:permission:visibility	Numerical
	Average denial rate for (A2) application$_F$:permission:visibility	Numerical

TABLE IV
THE COMPLETE LIST OF FEATURES USED IN THE ML MODEL EVALUATION. ALL THE NUMERICAL VALUES IN THE BEHAVIORAL GROUP ARE NORMALIZED PER DAY. WE USE ONE-HOT ENCODING FOR CATEGORICAL VARIABLES. WE NORMALIZED NUMERICAL VARIABLES BY MAKING EACH ONE A Z-SCORE RELATIVE TO ITS OWN AVERAGE.

unlocking, the proportion of times they allowed the screen to timeout instead of pressing the lock button, and the average amount of time spent unlocking the screen. Features under the audio and call category were the frequency of audio calls, the amount of time they spend on audio calls, and the proportion of time they spent on silent mode.

Our runtime features (*R1*/*R2*) include the requesting application's visibility, permission requested, and time of day of the request. Initially, we included the user ID to account for user-to-user variance, but as we discuss later, we subsequently removed it. Surprisingly, the application requesting the permission was not predictive, nor were other features based on the requesting application, such as application popularity.

Different users may have different ways of perceiving privacy threats posed by the same permission request. To account for this, the learning algorithm should be able to determine how each user perceives the appropriateness of a given request in order to accurately predict future decisions. To quantify the difference between users in how they perceive the threat posed by the same set of permission requests, we introduced a set of *aggregate features* that could be measured at runtime and that may partly capture users' privacy preferences. We compute the average denial rate for each unique combination of *application:permission:visibility* (*A1*) and of *application*$_F$ [5]*:permission:visibility* (*A2*). These aggregate features indicate how the user responded to previous prompts associated with that combination. As expected, after

[5]The application running in the foreground when the permission is requested by another application.

Feature Set	Contextuals	Defaulters	Overall
R1	69.30%	95.80%	83.71%
R2 + B	69.48%	95.92%	83.93%
R2 + A	75.45%	99.20%	92.24%

TABLE V
THE MEDIAN ACCURACY OF THE MACHINE LEARNING MODEL FOR DIFFERENT FEATURE GROUPS ACROSS DIFFERENT SUB POPULATIONS.

we introduced the aggregate features, the relative importance of the user ID variable diminished and so we removed it (i.e., users no longer needed to be uniquely identified). We define *R2* as *R1* without the user ID.

B. Inference Based on Behavior

One of our main hypotheses is that passively observing users' behaviors helps infer users' future privacy decisions. To this end, we instrumented Android to collect a wide array of behavioral data, listed in Table II. We categorize our behavioral instrumentation into interaction with Android privacy/security settings, locking behavior, audio settings and call habits, web-browsing habits, and application usage habits. After the feature selection process (§VI-A), we found that only locking behavior, audio habits, and web-browsing habits correlated with privacy behaviors. Appendix B contains more information on feature importance. All the numerical values under the behavioral group were normalized per day.

We trained an SVM model with an RBF kernel on only the behavioral and runtime features listed in Table IV, excluding user ID. The 5-fold cross-validation accuracy (with random splitting) was 83% across all users. This first setup assumes we have prior knowledge of previous privacy decisions to a certain extent from each user before inferring their future privacy decisions, so it is primarily relevant after the user has been using their phone for a while. However, the biggest advantage of using behavioral data is that it can be observed passively without any active user involvement (i.e., no prompting).

We use leave-one-out cross validation to measure the extent to which we can infer user privacy decisions with *absolutely no user involvement* (and without any prior data on a user). In this second setup, when a new user starts using a smartphone, we assume there is a ML model which is already trained with behavioral data and privacy decisions collected from a selected set of other users. We then measured the efficacy of such a model to predict the privacy decisions of a new user, purely based on passively observed behavior and runtime information on the request, without ever prompting that new user. This is an even stricter lower bound on user involvement, which essentially mandates that a user has to make no effort to indicate privacy preferences, something that no system currently does.

We performed leave-one-out cross validation for each of our 131 participants, meaning we predicted a single user's privacy decisions using a model trained using the data from the other 130 users' privacy decisions and behavioral data. The only input for each test user was the passively observed

behavioral data and runtime data surrounding each request. The model yielded a median accuracy of 75%, which is a 3 improvement over AOI. Furthermore, AOI requires users to make active decisions during the installation of an application, which our second model does not require.

Examining only behavioral data with leave-one-group-out cross validation yielded a median accuracy of 56% for *Contextuals*, while for *Defaulters* it was 93.01%. Although, prediction using solely behavioral data fell short of AOFU-AP for *Contextuals*, it yielded a similar median accuracy for *Defaulters*; AOFU-AP required 12 prompts to reach this level of accuracy, whereas our model would not have resulted in any prompts. This relative success presents the significant observation that behavioral features, observed passively without user involvement, are useful in learning user privacy preferences. This provides the potential to open entirely new avenues of user learning and reduce the risk of habituation.

C. Inference Based on Contextual Cues

Our SVM model with an RBF kernel produced the best accuracy. The results in the remainder of this section are trained and tested with five-fold cross validation with random splitting for an SVM model with an RBF kernel using the *ksvm* library in R. In all instances, the training set was bootstrapped with an equal number of allow and deny data points to avoid training a biased model. For each feature group, all hyperparameters were tuned through grid search to achieve highest accuracy. We used one-hot encoding for categorical variables. We normalized numerical variables by making each one a z-score relative to its own average. Table V shows how the median accuracy changes with different feature groups. As a minor note, the addition of the mentioned behavioral features to runtime features performed only marginally better; this could be due to the fact that those two groups do not complement each other in predictions. In this setup, we assume that there is a single model across all the users of Android.

By incorporating user involvement in the form of prompts, we can use our aggregate features to increase the accuracy for *Contextuals*, slightly less so for *Defaulters*. The aggregate features primarily capture how consistent users are for particular combinations (i.e., *application:permission:visibility*, *application$_F$:permission:visibility*), which greatly affects accuracy for *Contextuals*. *Defaulters* have high accuracy with just runtime features (*R1*), as they are likely to stick with a default allow or deny policy regardless of the context surrounding a permission. Thus, even without any aggregate features (which do not impart any new information about this type of user), the model can predict privacy preferences of *Defaulters* with a high degree of accuracy. On the other hand, *Contextuals* are more likely to vary their decision for a given permission request. However, as the accuracy numbers in Table V suggest, this variance is correlated with some contextual cues. The high predictive power of aggregate features indicates that they may be capturing the contextual cues, used by *Contextuals* to make decisions, to a greater extent.

The fact that both *application:permission:visibility* and *application$_F$:permission:visibility* are highly predictive (Appendix A) indicates that user responses for these combinations are consistent. The high consistency could relate to the notion that the visibility and the foreground application (application$_F$[6]) are strong contextual cues people use to make their privacy decisions; the only previously studied contextual cue was the visibility of the application requesting the sensitive data [43]. We offer a hypothesis for why foreground application could be significant: the sensitivity of the foreground application (i.e., high-sensitivity applications like banking, low-sensitivity applications like games) might impact how users perceive threats posed by requests. Irrespective of the application requesting the data, users may be likely to deny the request because of the elevated sense of risk. We discuss this further in §IX.

The model trained on feature sets *R2*, *A1*, and *A2* had the best accuracy (and the fewest privacy violations). For the remainder of the paper, we will refer to this model unless otherwise noted. We now compare AOFU-AP (the status quo as of Android 6.0 and above, presented in Table III) and our model (Table V). Across all users, our model reduced the error rate from 15.38% to 7.76%, nearly a two-fold improvement.

Mispredictions (errors) in the ML model were split between privacy violations and functionality losses (54% and 46%). Deciding which error type is more acceptable is subjective and depends on factors like the usability issues surrounding functionality losses and gravity of privacy violations. However, the (approximately) even split between the two error types shows that the ML is not biased towards one particular decision (denying vs. allowing a request). Furthermore, the area under the ROC curve (AUC), a metric used to measure the fairness of a classifier, is also significantly better in the ML model (0.936 as opposed to 0.796 for AOFU). This indicates that the ML model is equally good at predicting when to both allow and deny a permission request, while AOFU tends to lean more towards one decision. In particular, with the AOFU policy, users would experience privacy violations for 10.01% of decisions, compared to just 4.2% with the ML model. Privacy violations are likely more costly to the user than functionality loss: denied data can always be granted at a later time, but disclosed data cannot be taken back.

While increasing the number of prompts improves classifier accuracy, it plateaus after reaching its maximum accuracy, at a point we call the *steady state*. For some users, the classifier might not be able to infer their privacy preferences effectively, regardless of the number of prompts. As a metric to measure the effectiveness of the ML model, we measure the confidence of the model in the decisions it makes, based on prediction class probabilities.[7] In cases where the confidence of the model

[6]Even when the requesting application is running visible to the user, the foreground application could still be different from the requesting application since the only visible cue of the requesting application could be a notification in the notification bar.

[7]To calculate the class probabilities, we used the KSVM library in R. It employs a technique proposed by Platt et al. [25] to produce a numerical value for each class's probability.

is below a certain threshold, the system should use a runtime prompt to ask the user to make an explicit decision. Thus, we looked into the prevalence of low-confidence predictions among the current predictions. With a 95% confidence interval, on average across five folds, low-confidence predictions accounted for less than 10% of all predictions. The remaining high-confidence predictions (90% of all predictions) had an average accuracy of 96.2%, whereas predictions with low confidence were only predicted with an average accuracy of 72%. §VII-B goes into this aspect in detail and estimates the rate at which users will see prompts in steady state.

The caveat in our ML model is that AOFU-AP only resulted in 12 prompts on average per user during the study, while our model averaged 24. The increased prompting stems from multiple prompts for the same combination of *application:permission:visibility*, whereas in AOFU, prompts are shown only once for each *application:permission* combination. During the study period, users on average saw 2.28 prompts per unique combination. While multiple prompts per combination help the ML model to capture user preferences under different contextual circumstances, it risks habituation, which may eventually reduce the reliability of the user responses.

The evaluation setup mentioned in the current section does not have a specific strategy to select the training set. It randomly splits the data set into the 5 folds and picks 4 out of 5 as the training set. In a real-world setup, the platform needs a strategy to carefully select the training set so that the platform can learn most of the user's privacy preferences with a minimum number of prompts. The next section presents an in-depth analysis on possible ways to reduce the number of prompts needed to train the ML model.

VII. Learning Strategy

This sections presents a strategy the platform can follow in the learning phase of a new user. The key objective of the learning strategy should be to learn the user's privacy preferences with minimal user involvement (prompts). Once the model reaches adequate training, we can use model decision confidence to analyze how the ML model performs for different users and examine the tradeoff between user involvement and accuracy. We also utilize the model's confidence on decisions to present a strategy that can further reduce model error through selective permission prompting.

A. Bootstrapping

The *bootstrapping* phase occurs when the ML model is presented with a new user about whom the model has no prior information. In this section, we analyze how the accuracy improves as we prompt the user. Since the model presented in §VI is a single model trained with data from all users, the ML model can still predict a new user's privacy decisions by leveraging the data collected on other users' preferences.

We measured the accuracy of the ML model as if it had to predict each user's prompt responses using a model trained using other users' data. Formally, this is called leave-one-out cross-validation, where we remove all the prompt responses from a single user. The training set contains all the prompt responses from 130 users and the test set is the prompt responses collected from the single remaining user. The model had a median accuracy of 66.6% (56.2% for *Contextuals*, 86.4% for *Defaulters*). Although this approach does not prompt new users, it falls short of AOFU. This no-prompt model behaves close to random guessing for *Contextuals* and significantly better for *Defaulters*. Furthermore, Wijesekera et al. found that individuals' privacy preferences varied a lot [43], suggesting that utilizing other users' decisions to predict decisions for a new user has limited effectiveness, especially for *Contextuals*; some level of prompting is necessary.

There are a few interesting avenues to explore when determining the optimal way to prompt the user in the learning phase. One option would be to follow the same weighted-reservoir sampling algorithm mentioned in §III-A. The algorithm is weighted by the frequency of each *application:permission:visibility* combination. The most frequent combination will have the highest probability of creating a permission prompt and after the given combination reaches a maximum of three prompts, the algorithm will no longer consider that combination for prompting, giving the second most frequent combination the new highest probability. Due to frequency-weighting and multiple prompts per combination, the weighted-reservoir sampling approach requires more prompts to cover a broader set of combinations. However, AOFU prompts only once per combination without frequency-weighting. This may be a useful strategy initially for a new user since it allows the platform to learn about the users' privacy preferences for a wide array of combinations with minimal user interaction.

To simulate such an approach, we extend the aforementioned no-prompt model (leave-one-out validation). In the no-prompt model, there was no overlap of users in the train and test set. In the new approach, the training set includes the data from other users as well as the new user's responses to the first occurrence of each unique combination of *application:permission:visibility*. The first occurrence of each unique combination simulates the AOFU-APV policy. That is, this model is bootstrapped using data from other users and then adopts the AOFU-APV policy to further learn the current user's preferences. The experiment was conducted using the same set of features mentioned in §VI-A (*R2 + A1 + A2* and an SVM with a RBF kernel). The test set only contained prompt responses collected after the last AOFU prompt to ensure chronological consistency.

Figure 3 shows how accuracy changes with the varying number of AOFU prompts for *Contextuals* and *Defaulters*. For each of the 131 users, we ran the experiment varying the AOFU prompts from 1 to 12. We chose this upper bound because, on average, a participant saw 12 different unique *application:permission* combinations during the study period—the current permission model in Android. AOFU relies on user prompts for each new combination. The proposed ML model, however, has the advantage of leveraging data collected from other users to predict a combination not seen by the user; it can

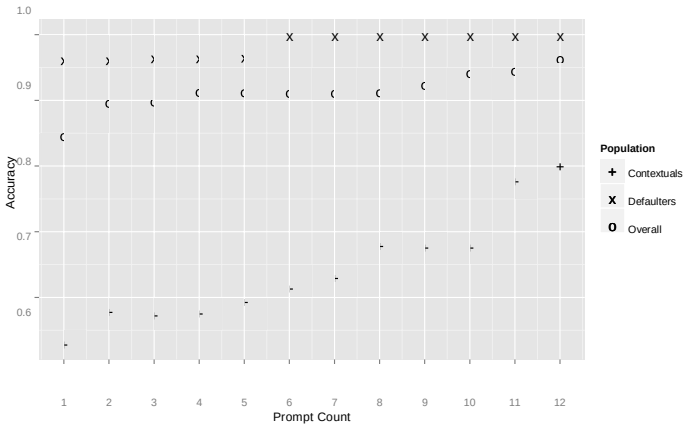


Fig. 3. How the median accuracy varies with the number of seen prompts

significantly reduce user involvement in the learning phase. After 12 prompts, accuracy reached 96.8% across all users.

Each new user starts off with a single model shared by all new users and then moves onto a separate model trained with AOFU prompt responses. We analyze its performance for *Defaulters* and *Contextuals* separately, finding that it improves accuracy while reducing user involvement in both cases, compared to the status quo.

We first examine how our model performs for *Defaulters*, 53% of our sample. Figure 3 shows that our model trained with AOFU permission-prompt responses outperforms AOFU from the very beginning. The model starts off with 96.6% accuracy (before it reaches close to 100% after 6 prompts), handily exceeding AOFU's 93.33%. This is a 83.3% reduction in permission prompts compared to AOFU-AP (the status quo). Even with such a significant reduction in user involvement, the new approach cuts the prediction error rate in half.

Contextuals needed more prompts to outperform the AOFU policy; the hybrid approach matches AOFU-AP with just 7 prompts, a 42% reduction in prompts. With 12 permission prompts, same as needed for AOFU-AP, the new approach had reduced the error rate by 43% over AOFU-AP (the status quo). The number of prompts needed to reach this level of accuracy in the new approach is 25% less than what is needed for AOFU-APV. We also observed that as the number of prompts increased, the AUC of our predictions also similarly increased. Overall, the proposed learning strategy reduced the error rate by 80% after 12 user prompts over AOFU-AP. Given, *Defaulters* plateau early in their learning cycle (after only 6 prompts), the proposed learning strategy, on average, needs 9 prompts to reach its maximum capacity, which is a 25% reduction in user involvement over AOFU-AP.

Contextuals have a higher need for user involvement than *Defaulters*, primarily because it is easy to learn about *Defaulters*, as they are more likely to be consistent with early decisions. On the other hand, *Contextuals* vary their decisions based on different contextual cues and require more user involvement for the model to learn the cues used by each user and how do they affect their decisions. Thus, it is important to find a way to differentiate between *Defaulters* and *Contextuals* early in the bootstrapping phase to determine which users require fewer prompts. The analysis of our hybrid approach addresses the concern of a high number of permission prompts initially for an ML approach. Over time, accuracy can always be improved with more prompts.

Our new hybrid approach of using AOFU-style permission prompts in the bootstrapping phase to train our model can achieve higher accuracy than AOFU, with significantly fewer prompts. Having a learning strategy (use of AOFU) over random selection helped to minimize user involvement (24 vs. 9) while significantly reducing the error rate (7.6% vs. 3.2%) over a random selection of the training set.

B. Decision Confidence

In the previous section, we looked into how we can optimize the learning phase by merging AOFU and the ML model to reach higher accuracy with minimal user prompts. However, for a small set of users, more permission prompts will not increase accuracy, regardless of user involvement in the boot- strapping phase. This could be due to the fact that a portion of users in our dataset are making random decisions, or that the features that our ML model takes into account are not predictive of those users' decision processes. While we do not have the data to support either explanation, we examine how we can measure whether the ML model will perform well for a particular user and quantify how often it does not. We present a method to identify difficult-to-predict users and reduce permission prompting for those users.

While running the experiment in §VII-A, we also measured how confident the ML model was for each decision it made. To measure the ML model's confidence, we record the probability for each decision; since it is a binary classification (deny or allow), the closer the probability is to 0.5, the less confident it is. We then chose a *class probability threshold* above which a decision would be considered a high-confidence decision. In our analysis, we choose a class probability threshold of 0.6, since this value resulted in >96% accuracy for our fully- trained model (25 prompts per user) for high-confidence decisions, but this is a tunable threshold. Thus, in the remainder of our analysis, decisions that the ML model made with a probability of >0.60 were labeled as high-confidence decisions, while those made with a probability of <0.60 were labeled as low-confidence decisions.

Since the most accurate version of AOFU uses 12 prompts, we also evaluate the confidence of our model after 12 AOFU-style prompts. This setup is identical to the bootstrapping approach; the model we evaluate here is trained on responses from other users and the first 12 prompts chosen by AOFU. With this scheme, we found that 10 users (7.63% of 131 users) had at least one decision predicted with low confidence. The remaining 92.37% of users had all privacy decisions predicted with high confidence. Among those users whose decisions were predicted with low confidence, the proportion of low-confidence decisions on average accounted for 17.63% (median = 16.67%) out of all their predicted decisions. With

a sensitive permission request once every 15 seconds [43], prompting even for 17.63% of predictions is not practical. Users who had low-confidence predictions had a median accuracy of 60.17%, compared to 98% accuracy for the remaining set of users with only high-confidence predictions. Out of the 10 users who had low-confidence predictions, there were no *Defaulters*. This further supports the observation in Figure 3 that *Defaulters* require a shorter learning period.

In a real-world scenario, after the platform (ML model) prompts the user for the first 12 AOFU prompts, the platform can measure the confidence of predicting unlabeled data (sensitive permission requests for which the platform did not prompt the user). If the proportion of low-confidence predictions is below some threshold, the ML model can be deemed to have successfully learned user privacy preferences and the platform should keep on using the regular permission-prompting strategy. Otherwise, the platform may choose to limit prompts (i.e., two per unique *application:permission:visibility* combination). It should also be noted that rather than having a fixed number of prompts (e.g., 12) to measure the low-confidence proportion, the platform can keep track of the low-confidence proportion as it prompts the user according to any heuristic (i.e., unique combinations). If the proportion does not decrease with the number of prompts, we can infer that the ML model is not learning user preferences effectively or the user is making random decisions, indicating that limiting prompts and accepting lower accuracy could be a better option for that specific user, to avoid excessive prompting. However, depending on which group the user is in (*Contextual* or *Defaulter*), the point at which the platform could make the decision to continue or limit prompting could change. In general, the platform should be able to reach this deciding point relatively quickly for *Defaulters.*

Among participants with no low-confidence predictions, we had a median error rate of 2% (using the new hybrid approach after 12 AOFU prompts); for the same set of users, AOFU could only reach a median error rate of 13.3%. However, using AOFU, a user in that set would have needed an average of 15.11 prompts to reach that accuracy. Using the ML model, a user would need just 9 prompts on average (*Defaulters* require far fewer prompts, dropping the average); the model only requires 60% of the prompts that AOFU requires. Even with far fewer prompts in the learning phase, the ML model achieves a 84.61% reduction in error rate relative to AOFU.

While our model may not perform well for all users, it does seem to work quite well for the majority of users (92.37% of our sample). We provide a way of quickly identifying users for whom our system does not perform well, and propose limiting prompts to avoid excessive user burden for those users, at the cost of reduced efficacy. In the worst case, we could simply employ the AOFU model for users our system does not work well for, resulting in a multifaceted approach that is at least as good as the status quo for all users.

C. Online Model

Our proposed system relies on training models on a trusted server, sending it to client phones (i.e., as a weight vector), and having phones make classifications. By utilizing an online learning model, we can train models incrementally as users respond to prompts over time. There are two key advantages to this: (i) this model adapts to changing user preferences over time; (ii) it distributes the overhead of training increasing the practicality of locally training the classifier on the phone itself. Our scheme requires two components: a feature extraction and storage mechanism on the phone (a small extension to our existing instrumentation) and a machine learning pipeline on a trusted server. The phone sends feature vectors to the server every few prompts, and the server responds with a weight vector representing the newly trained classifier. To bootstrap the process, the server's models can be initialized with a model trained on a few hundred users, such as our single model across all users. Since each user contributes data points over time, the online model adapts to changing privacy preferences even if they conflict with previous data. When using this scheme, each model takes less than 10 KB to store. With our current model, each feature and weight vector are at most 3 KB each, resulting in at most 6 KB of data transfer per day.

To evaluate the accuracy of our online model, we trained a classifier using stochastic gradient descent (SGD) with five-fold cross validation on our 4,224-point data set. This served as the bootstrapping phase. We then simulated receiving the remaining data one-at-a-time in timestamp order. Any features that changed with time (e.g., running averages for aggregate features, event counts) were computed with each incoming data point, creating a snapshot of features as the phone would see it. We then tested accuracy on the chronologically last 20% of our dataset. Our SGD classifier had 93.8% accuracy (AUC=0.929). We attribute the drop in accuracy (compared to our offline model) to the fact that running averages take multiple data points to reach steady-state, causing some earlier predictions to be incorrect.

A natural concern with a trusted server is compromise. To address this concern, we do not send any personally-identifiable data to the server, and any features sent to the server are *scaled*; they are reported in standard deviations from the mean, not in raw values. Furthermore, using an online model with incremental training allows us to periodically train the model on the phone (i.e., nightly, when the user is charging her device) to eliminate the need for a trusted server.

VIII. Contextual Integrity

Contextual integrity is a conceptual framework that helps explain why most permission models fail to protect user privacy—they often do not take the context surrounding privacy decisions into account. In addressing this issue, we propose an ML model that infers when context has changed. We believe that this is an important first step towards operationalizing the notion of *contextual integrity*. In this section, we explain the observations that we made in §VI-C based on the contextual integrity framework proposed by Barth et al. [6].

Contextual integrity provides a conceptual framework to better understand how users make privacy decisions; we use Barth et al.'s formalized model [6] as a framework in which to view Android permission models. Barth et al. model parties as communicating agents (P) knowing information represented as attributes (T). A knowledge state κ is defined as a subset of $P \rightarrow P \rightarrow T$. We use $\kappa = (p, q, t)$ to mean that agent p knows attribute t of agent q. Agents play roles (R) in contexts (C). For example, an agent can be a game application, and have the role of a game provider in an entertainment context. Knowledge transfer happens when information is communicated between agents; all communications can be represented through a series of traces $(\kappa, (p, r), a)$, which are combinations of a knowledge state κ, a role state (p, r), and a communication action a (information sent). The role an agent plays in a given context helps determine whether an information flow is acceptable for a user. The relationship between the agent sending the information and the role of the agent $((p, r))$ receiving the information must follow these contextual norms. With the Android permission model, the same framework can be applied. Both the user and the third-party application are communicating agents, and the information to be transferred is the sensitive data requested by the application. When a third-party application requests permission to access a guarded resource (e.g., location), knowledge of the guarded resource is transferred from the one agent (i.e., the user/platform) to another agent (i.e., the third-party application). The extent to which a user expects a given request depends not on the agent (the application requesting the data), but on the role that agent is playing in that context. This explains why the application as a feature itself (i.e., application name) was not predictive in our models: this feature does not represent the role when determining whether it is unexpected. While it is difficult for the platform to determine the exact role an application is playing, the visibility of the application hints at its role. For instance, when the user is using Google Maps to navigate, it is playing a different role from when Google Maps is running in the background without the user's knowledge. We believe that this is the reason why the visibility of the requesting application is significant: it helps the user to infer the role played by the application requesting the permission. The user expects applications in certain roles to access resources depending on the context in which the request is made. We believe that the foreground application sets this context. Thus a combination of the role and the context decides whether an information flow is expected to occur or not. Automatically inferring the exact context of a request is likely an intractable problem. For our purposes, however, it is possible that we need to only infer when context has *changed*, or rather, when data is being requested in a context that is no longer acceptable to the user. Based on our data, we believe that features based on foreground application and visibility are most useful for this purpose, from our collected dataset.

We now combine all of this into a concrete example within the contextual integrity framework: If a user is using Google Maps to reach a destination, the application can play the role of a navigator in a geolocation context, whereby the user feels comfortable sharing her location. In contrast, if the same application requests location while running as a service invisible to the user, the user may not want to provide the same information. Background applications play the role of "passive listeners" in most contexts; this role as perceived by the user may be why background applications are likelier to violate privacy expectations and consequently be denied by users.

AOFU primarily focuses on controlling access through rules for *application:permission* combinations. Thus, AOFU neglects the role played by the application (visibility) and relies purely on the agent (the application) and the information subject (permission type). This explains why AOFU is wrong in nearly one-fifth of cases. Based on Table III, both AOFU- VA (possibly identifying the role played by the application) and AOFU-A_FPV (possibly identifying the current context because of the current foreground application-A_F) have higher accuracy than the other AOFU combinations. However, as the contextual integrity framework suggests, the permission model has to take both the role and the current context into account before making an accurate decision. AOFU (and other models that neglect context) only makes it possible to consider a single aspect, a limitation that does not apply to our model.

While the data presented in this work suggest the importance of capturing context to better protect user privacy, more work is needed along these lines to fully understand how people use context to make decisions in the Android permission model. Nevertheless, we believe we contribute a significant initial step towards applying contextual integrity to improve smartphone privacy by dynamically regulating permissions.

IX. Discussion

The primary goal of this research was to improve the accuracy of the Android permission system so that it more correctly aligns with user privacy preferences. We began with four hypotheses: (i) that the currently deployed AOFU policy frequently violates user privacy; (ii) that the contextual information it ignores is useful; (iii) that a ML-based classifier can account for this contextual information and thus improve on the status quo; and (iv) that passively observable behavioral traits can be used to infer privacy preferences.

To test these hypotheses, we performed the first large-scale study on the effectiveness of AOFU permission systems in the wild, which showed that hypotheses (i) and (ii) hold. We further built an ML classifier that took user permission decisions along with observations of user behaviors and the context surrounding those decisions to show that (iii) and (iv) hold. Our results show that existing systems have significant room for improvement, and other permission-granting systems may benefit from applying our results.

A. Limitations of Permission Models

Our field study confirms that users care about their privacy and are wary of permission requests that violate their expectations. We observed that 95% of participants chose to block at least one permission request; in fact, the average denial rate was

60%—a staggering amount given that the AOI model permits all permission requests for an installed application.

While AOFU improves over the AOI model, it still violates user privacy around one in seven times, as users deviate from their initial responses to permission requests. This amount is significant because of the high frequency of sensitive permission requests: a 15% error rate yields thousands of privacy violations per user—based on the latest dataset, this amounts to a potential privacy violation every minute. It further shows that AOFU's correctness assumption—that users make binary decisions based only on the *application:permission* combination—is incorrect. Users take a richer space of information into account when making decisions about permission requests.

B. Our ML-Based Model

We show that ML techniques are effective at learning from both the user's previous decisions and the current environmental context in order to predict whether to grant permissions on the user's behalf. In fact, our techniques achieve better results than the methods currently deployed on millions of phones worldwide—while imposing significantly less user burden.

Our work incorporates elements of the surrounding context into a machine-learning model. This better approximates user decisions by finding factors relevant for users that are not encapsulated by the AOFU model. In fact, our ML model reduces the errors made by the AOFU model by 75%. Our ML model's 97% accuracy is a substantial improvement over AOFU's 85% and AOI's 25%; the latter two of which comprise the *status quo* in the Android ecosystem.

Our research shows that many users make neither random nor fixed decisions: the environmental context plays a significant role in user decision-making. Automatically detecting the precise context surrounding a request for sensitive data is an incredibly difficult problem (e.g., inferring *how* data will be used), and is potentially intractable. However, to better support user privacy, that problem does not need to be solved; instead, we show that systems can be improved by using environmental data to infer when context has *changed*. We found that the most predictive factors in the environmental context were whether the application requesting the permission is visible, and what the foreground application the user is engaged with. These are both strong contextual cues used by users, insofar as they allowed us to better predict changes in context. Our results show that ML techniques have great potential in improving user privacy, by allowing us to infer when context has changed, and therefore when users would want data requests to be brought to their attention.

C. Reducing the User Burden

Our work is also novel in using passively observable data to infer privacy decisions: we show that we can predict a user's preferences without *any* permission prompts. Our model trained solely on behavioral traits yields a three-fold improvement over AOI; for *Defaulters*—who account for 53% of our sample—it was as accurate as AOFU-AP. These results demonstrate that we can match the status quo without *any* active user involvement (i.e., the need for obtrusive prompts). These results imply that learning privacy preferences may be done entirely passively, which, to our knowledge, has not yet been attempted in this domain. Our behavioral feature set provides a promising new direction to guide research in creating permission models that minimize user burden.

The ML model trained with contextual data and past decisions also significantly reduced the user burden while achieving higher accuracy than AOFU. The model yielded an 81% reduction in prediction errors while reducing user involvement by 25%. The significance of this observation is that by reducing the risk of habituation, it increases reliability when user input is needed.

D. User- and Permission-Tailored Models

Our ML-based model incorporates data from all users into a single predictive model. It may be the case, however, that a collection of models tailored to particular types of users outperforms our general-purpose model—provided that the correct model is used for the particular user and permission. To determine if this is true, we clustered users into groups based first on their behavioral features, and then their denial rate, to see if we could build superior cluster-tailored ML models. Having data for only 131 users, however, resulted in clusters too small to carry out an effective analysis. We note that we also created a separate model for each sensitive permission type, using data only for that permission. Our experiments determined, however, that these models were no better (and often worse) than our general model. It is possible that such tailored models may be more useful when our system is implemented at scale.

E. Attacking the ML Model

Attacking the ML model to get access to users' data without prompting is a legitimate concern [5]. There are multiple ways an adversary can influence the proposed permission model: (i) imposing an adversarial ML environment [31]; (ii) polluting the training set to bias the model to accept permissions; and (iii) manipulating input features in order to get access without user notification. We assume in this work that the platform is not compromised; a compromised platform will degrade any permission model's ability to protect resources.

A thorough analysis on this topic is outside of our scope. Despite that, we looked at the possibility of manipulating features to get access to resources without user consent. None of the behavioral features used in the model can be influenced, since that would require compromising the platform. An adversary can control the runtime features for a given permission request by specifically choosing when to request the permission. We generated feature vectors manipulating every adversary-controlled value and combination from our dataset, and tested them on our model. We did not find any conclusive evidence that the adversary can exploit the ML model by manipulating the input features to get access to resources without user consent.

As this is not a comprehensive analysis on attack vectors, it is possible that a scenario exists where the adversary is able to access sensitive resources without prompting the user first. Our preliminary analysis suggests that such attacks may be non-trivial, but more work is needed to study and prevent such attacks, particularly examining adversarial ML techniques and feature brittleness.

F. Experimental Caveat

We repeat a caveat about our experimental data: users were free to deny permissions without any consequences. We explicitly informed participants in our study that their decisions to deny permission requests would have no impact on the actual behavior of their applications. This is important to note because if an application is denied a permission, it may exhibit undefined behavior or lose important functionality. In fact, researchers have noted that many applications crash when permissions are denied [13]. If these consequences are imposed on users, they may decide that the functionality is more important than their privacy decision.

If we actually denied permissions, users' decisions may skew towards a decreased denial rate. The denial rates in our experiments therefore represent the actual privacy preferences of users and their *expectations* of reasonable application behavior—not the result of choosing between application functionality and privacy. We believe that how people react when choosing between functionality and privacy preferences is an important research question beyond the scope of this paper. Such a change, however, will not limit this contribution, since our proposed model was effective in guarding resources of the users who are selective in their decision making—the proposed classifier reduced the error rate of *Contextuals* by 44%.

We believe that there are important unanswered questions about how to solve the technical hurdles surrounding enforcing restrictive preferences with minimal usability issues. As a first step towards building a platform that does not force users to choose between their privacy preferences and required functionality, we must develop an environment where permissions appear—to the application—to be allowed, but in reality only spurious or artificial data is provided.

G. Types of Users

We presented a categorization of users based on the significance that the application's visibility played towards their individual privacy decisions. We believe that in an actual permission denial setting, the distribution will be different from what was observed in our study. Our categorization's significance, however, motivates a deeper analysis on understanding the factors that divide *Contextuals* and *Defaulters*. While visibility was an important factor in this division, there may be others that are significant and relevant. More work needs to be done to explore how *Contextuals* make decisions and which behaviors correlate with their decisions.

H. User Interface Panel

Any model that predicts user decisions has the risk of making incorrect predictions. Making predictions on a user's behalf, however, is necessary because permissions are re- quested by applications with too high a frequency for manual examination. While we do not expect any system to be able to obtain perfect accuracy, we do expect that our 97% accuracy can be improved upon.

One plausible way of improving the accuracy of the permission model is to empower the user to review and make changes on how the ML model makes decisions through a user feedback panel. This gives users recourse to correct undesirable decisions. The UI panel could also be used to reduce the usability issues and functionality loss stemming from permission denial. The panel should help the user figure out which rule incurred the functionality loss and to change it accordingly. A user may also use this to adjust their settings as their privacy preferences evolve over time.

I. The Cost of Greater Control

A more restrictive platform means users will have greater control over the data being shared with third parties. Applications that generate revenue based on user data, however, could be cut off from their primary revenue source. Such an effect could disrupt the current eco-system and force app developers to degrade app functionality based on the availability of the data. We believe the current eco-system is unfairly biased against users and tighter control will make the user an equal stakeholder. While more work is needed to understand the effects of a more restrictive platform, we believe it is imperative to let the user have greater control over their own data.

J. Conclusions

We have shown a number of important results. Users care about their privacy: they deny a significant number of requests to access sensitive data. Existing permission models for Android phones still result in significant privacy violations. Users may allow permissions sometimes, while denying them at others, implying that there are more factors that go into the decision-making process than simply the application name and the permission type. We collected real-world data from 131 users and found that application visibility and the current foreground application were important factors in user decisions. We used the data we collected to build a machine-learning model to make automatic permission decisions. One of our models had a comparable error rate to AOFU and benefited from not requiring any user prompting. Another of our models required some user prompts—less than is required by AOFU— and achieved a reduction of AOFU's error rate by 81%.

ACKNOWLEDGMENTS

This research was supported by the United States Department of Homeland Security's Science and Technology Directorate under contract FA8750-16-C-0140, the Center for Long-Term Cybersecurity (CLTC) at UC Berkeley, the National Science Foundation under grant CNS-1318680, and Intel through the ISTC for Secure Computing. The content of this document does not necessarily reflect the position or the policy of the U.S. Government and no official endorsement should be inferred.

References

[1] Y. Agarwal and M. Hall, "Protectmyprivacy: Detecting and mitigating privacy leaks on ios devices using crowdsourcing," in *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '13. New York, NY, USA: ACM, 2013, pp. 97–110. [Online]. Available: http://doi.acm.org/10.1145/2462456.2464460

[2] H. M. Almohri, D. D. Yao, and D. Kafura, "Droidbarrier: Know what is executing on your android," in *Proc. of the 4th ACM Conf. on Data and Application Security and Privacy*, ser. CODASPY '14. New York, NY, USA: ACM, 2014, pp. 257–264. [Online]. Available: http://doi.acm.org/10.1145/2557547.2557571

[3] H. Almuhimedi, F. Schaub, N. Sadeh, I. Adjerid, A. Acquisti, J. Gluck, L. F. Cranor, and Y. Agarwal, "Your location has been shared 5,398 times!: A field study on mobile app privacy nudging," in *Proc. of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 2015, pp. 787–796.

[4] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "Pscout: Analyzing the android permission specification," in *Proc. of the 2012 ACM Conf. on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 217–228. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382222

[5] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, "Can machine learning be secure?" in *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*. ACM, 2006, pp. 16–25.

[6] A. Barth, A. Datta, J. C. Mitchell, and H. Nissenbaum, "Privacy and contextual integrity: Framework and applications," in *Proc. of the 2006 IEEE Symposium on Security and Privacy*, ser. SP '06. Washington, DC, USA: IEEE Computer Society, 2006. [Online]. Available: http://dx.doi.org/10.1109/SP.2006.32

[7] I. Bilogrevic, K. Huguenin, B. Agir, M. Jadliwala, and J.-P. Hubaux, "Adaptive information-sharing for privacy-aware mobile social networks," in *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, ser. UbiComp '13. New York, NY, USA: ACM, 2013, pp. 657–666. [Online]. Available: http://doi.acm.org/10.1145/2493432.2493510

[8] E. Bodden, "Easily instrumenting android applications for security purposes," in *Proc. of the ACM Conf. on Comp. and Comm. Sec.*, ser. CCS '13. NY, NY, USA: ACM, 2013, pp. 1499–1502. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516759

[9] A. Developer, "Requesting permissions," https://developer.android.com/guide/topics/permissions/requesting.html, accessed: March 18, 2017.

[10] G. Developer, "Distribution of android versions," http://developer.android.com/about/dashboards/index.html, accessed: March 15, 2017.

[11] S. Egelman, A. P. Felt, and D. Wagner, "Choice architecture and smartphone privacy: There's a price for that," in *The 2012 Workshop on the Economics of Information Security (WEIS)*, 2012.

[12] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924971

[13] Z. Fang, W. Han, D. Li, Z. Guo, D. Guo, X. S. Wang, Z. Qian, and H. Chen, "revdroid: Code analysis of the side effects after dynamic permission revocation of android apps," in *Proceedings of the 11th ACM Asia Conference on Computer and Communications Security (ASIACCS 2016)*. Xi'an, China: ACM, 2016.

[14] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proc. of the ACM Conf. on Comp. and Comm. Sec.*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 627–638. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046779

[15] A. P. Felt, S. Egelman, M. Finifter, D. Akhawe, and D. Wagner, "How to ask for permission," in *Proc. of the 7th USENIX conference on Hot Topics in Security*. Berkeley, CA, USA: USENIX Association, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2372387.2372394

[16] A. P. Felt, S. Egelman, and D. Wagner, "I've got 99 problems, but vibration ain't one: a survey of smartphone users' concerns," in *Proc. of the 2nd ACM workshop on Security and Privacy in Smartphones and Mobile devices*, ser. SPSM '12. New York, NY, USA: ACM, 2012, pp. 33–44. [Online]. Available: http://doi.acm.org/10.1145/2381934.2381943

[17] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: user attention, comprehension, and behavior," in *Proc. of the Eighth Symposium on Usable Privacy and Security*, ser. SOUPS '12. New York, NY, USA: ACM, 2012. [Online]. Available: http://doi.acm.org/10.1145/2335356.2335360

[18] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Androidleaks: Automatically detecting potential privacy leaks in android applications on a large scale," in *Proc. of the 5th Intl. Conf. on Trust and Trustworthy Computing*, ser. TRUST'12. Berlin, Heidelberg: Springer- Verlag, 2012, pp. 291–307. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-30921-2_17

[19] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 1025–1035. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568276

[20] S. E. Hormuth, "The sampling of experiences in situ," *Journal of personality*, vol. 54, no. 1, pp. 262–293, 1986.

[21] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, "These aren't the droids you're looking for: retrofitting android to protect data from imperious applications," in *Proc. of the ACM Conf. on Comp. and Comm. Sec.*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 639–652. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046780

[22] J. Jung, S. Han, and D. Wetherall, "Short paper: Enhancing mobile application permissions with runtime feedback and constraints," in *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, ser. SPSM '12. New York, NY, USA: ACM, 2012, pp. 45–50. [Online]. Available: http://doi.acm.org/10.1145/2381934.2381944

[23] P. G. Kelley, S. Consolvo, L. F. Cranor, J. Jung, N. Sadeh, and D. Wetherall, "A conundrum of permissions: Installing applications on an android smartphone," in *Proc. of the 16th Intl. Conf. on Financial Cryptography and Data Sec.*, ser. FC'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 68–79. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-34638-5_6

[24] W. Klieber, L. Flynn, A. Bhosale, L. Jia, and L. Bauer, "Android taint flow analysis for app sets," in *Proceedings of the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis*, ser. SOAP '14, New York, NY, USA, 2014. [Online]. Available: http://doi.acm.org/10.1145/2614628.2614633

[25] H.-T. Lin, C.-J. Lin, and R. C. Weng, "A note on platt's probabilistic outputs for support vector machines," *Machine learning*, vol. 68, no. 3, pp. 267–276, 2007.

[26] J. Lin, B. Liu, N. Sadeh, and J. I. Hong, "Modeling users' mobile app privacy preferences: Restoring usability in a sea of permission settings," in *Symposium On Usable Privacy and Security (SOUPS 2014)*. Menlo Park, CA: USENIX Association, 2014, pp. 199–212. [Online]. Available: https://www.usenix.org/conference/soups2014/proceedings/presentation/lin

[27] J. Lin, N. Sadeh, S. Amini, J. Lindqvist, J. I. Hong, and J. Zhang, "Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing," in *Proc. of the 2012 ACM Conf. on Ubiquitous Computing*, ser. UbiComp '12. New York, NY, USA: ACM, 2012, pp. 501–510. [Online]. Available: http://doi.acm.org/10.1145/2370216.2370290

[28] B. Liu, M. S. Andersen, F. Schaub, H. Almuhimedi, S. A. Zhang, N. Sadeh, Y. Agarwal, and A. Acquisti, "Follow my recommendations: A personalized assistant for mobile app permissions," in *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, 2016.

[29] B. Liu, J. Lin, and N. Sadeh, "Reconciling mobile app privacy and usability on smartphones: Could user privacy profiles help?" in *Proceedings of the 23rd International Conference on World Wide Web*, ser. WWW '14. New York, NY, USA: ACM, 2014, pp. 201–212. [Online]. Available: http://doi.acm.org/10.1145/2566486.2568035

[30] G. Louppe, L. Wehenkel, A. Sutera, and P. Geurts, "Understanding variable importances in forests of randomized trees," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013. [Online]. Available: http://papers.nips.cc/paper/4928-understanding-variable-importances-in-forests-of-randomized-trees.pdf

[31] D. Lowd and C. Meek, "Adversarial learning," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 2005, pp. 641–647.
[32] K. Micinski, D. Votipka, R. Stevens, N. Kofinas, J. S. Foster, and M. L. Mazurek, "User interactions and permission use on android," in *CHI 2017*, 2017.
[33] A. Nandugudi, A. Maiti, T. Ki, F. Bulut, M. Demirbas, T. Kosar, C. Qiao, S. Y. Ko, and G. Challen, "Phonelab: A large programmable smartphone testbed," in *Proceedings of First International Workshop on Sensing and Big Data Mining*. ACM, 2013, pp. 1–6.
[34] H. Nissenbaum, "Privacy as contextual integrity," *Washington Law Review*, vol. 79, p. 119, February 2004.
[35] T. Ringer, D. Grossman, and F. Roesner, "Audacious: User-driven access control with unmodified operating systems," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 204–216.
[36] F. Roesner and T. Kohno, "Securing embedded user interfaces: Android and beyond," in *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, 2013, pp. 97–112.
[37] F. Roesner, T. Kohno, A. Moshchuk, B. Parno, H. J. Wang, and C. Cowan, "User-driven access control: Rethinking permission granting in modern operating systems," in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 224–238.
[38] J. L. B. L. N. Sadeh and J. I. Hong, "Modeling users' mobile app privacy preferences: Restoring usability in a sea of permission settings," in *Symposium on Usable Privacy and Security (SOUPS)*, 2014.
[39] B. Shebaro, O. Oluwatimi, D. Midi, and E. Bertino, "Identidroid: Android can finally wear its anonymous suit," *Trans. Data Privacy*, vol. 7, no. 1, pp. 27–50, Apr. 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2612163.2612165
[40] M. Spreitzenbarth, F. Freiling, F. Echtler, T. Schreck, and J. Hoffmann, "Mobile-sandbox: Having a deeper look into android applications," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, ser. SAC '13. New York, NY, USA: ACM, 2013. [Online]. Available: http://doi.acm.org/10.1145/2480362.2480701
[41] C. Thompson, M. Johnson, S. Egelman, D. Wagner, and J. King, "When it's better to ask forgiveness than get permission: Designing usable audit mechanisms for mobile permissions," in *Proc. of the 2013 Symposium on Usable Privacy and Security (SOUPS)*, 2013.
[42] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos, "Permission evolution in the android ecosystem," in *Proceedings of the 28th Annual Computer Security Applications Conference*, ser. ACSAC '12. New York, NY, USA: ACM, 2012, pp. 31–40. [Online]. Available: http://doi.acm.org/10.1145/2420950.2420956
[43] P. Wijesekera, A. Baokar, A. Hosseini, S. Egelman, D. Wagner, and K. Beznosov, "Android permissions remystified: A field study on contextual integrity," in *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, Aug. 2015, pp. 499–514. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/wijesekera
[44] H. Wu, B. P. Knijnenburg, and A. Kobsa, "Improving the prediction of users' disclosure behavior by making them disclose more predictably?" in *Symposium on Usable Privacy and Security (SOUPS)*, 2014.
[45] K.-P. Yee, "Guidelines and strategies for secure interaction design," *Security and Usability: Designing Secure Systems That People Can Use*, vol. 247, 2005.
[46] H. Zhu, H. Xiong, Y. Ge, and E. Chen, "Mobile app recommendations with security and privacy awareness," in *Proc. of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2014. [Online]. Available: http://doi.acm.org/10.1145/2623330.2623705

APPENDIX A
INFORMATION GAIN OF CONTEXTUAL FEATURES

	Contextuals	Defaulters	Overall
A1	0.4839	0.6444	0.5717
A2	0.4558	0.6395	0.5605
Permission	0.0040	0.0038	0.0050
Time	0.0487	0.1391	0.0130
Visibility	0.0015	0.0007	0.0010

TABLE VI
FEATURE IMPORTANCE OF CONTEXTUAL FEATURES

APPENDIX B
INFORMATION GAIN OF BEHAVIORAL FEATURES

Feature	**Importance**
Amount of time spent on audio calls	0.327647825
Frequency of audio calls	0.321291184
Proportion of times screen was timed out instead of pressing the lock button	0.317631096
Number of times PIN was used to unlock the screen.	0.305287288
Number of screen unlock attempts	0.299564131
Amount of time spent unlocking the screen	0.29930659
Proportion of time spent on loud mode	0.163166296
Proportion of time spent on silent mode	0.138469725
Number of times a website is loaded to the Chrome browser	0.094996437
Out of all visited websites, the proportion of HTTPS-secured websites.	0.071096898
Number of times Password was used to unlock the screen	0.067999523
Proportion of websites requested location through Chrome	0.028404167
Time	0.019799623
The number of downloads through Chrome	0.014619351
Permission	0.001461635
Visibility	0.000162166

TABLE VII
FEATURE IMPORTANCE OF BEHAVIORAL FEATURES

Appendix C TurtleGuard: Helping Android Users Apply Contextual Privacy Preferences

TurtleGuard: Helping Android Users Apply Contextual Privacy Preferences

Lynn Tsai[1], Primal Wijesekera[2], Joel Reardon[1], Irwin Reyes[3], Jung-Wei Chen[4], Nathan Good[4], Serge Egelman[1,3], and David Wagner[1]

[1]University of California, Berkeley, Berkeley, CA
{lynntsai,daw}@cs.berkeley.edu, jreardon@berkeley.edu

[2]University of British Columbia, Vancouver, BC
primal@ece.ubc.ca

[3]International Computer Science Institute, Berkeley, CA
{ioreyes,egelman}@icsi.berkeley.edu

[4]Good Research, Inc., El Cerrito, CA
{jennifer,nathan}@goodresearch.com

ABSTRACT

Current mobile platforms provide privacy management interfaces to regulate how applications access sensitive data. Prior research has shown how these interfaces are insufficient from a usability standpoint: they do not account for *context*. In allowing for more contextual decisions, machine-learning techniques have shown great promise for designing systems that automatically make privacy decisions on behalf of the user. However, if such decisions are made automatically, then feedback mechanisms are needed to empower users to both audit those decisions and correct any errors.

In this paper, we describe our user-centered approach towards designing a fully functional privacy feedback interface for the Android platform. We performed two large-scale user studies to research the usability of our design. Our second, 580-person validation study showed that users of our new interface were significantly more likely to both understand and control the selected set of circumstances under which applications could access sensitive data when compared to the default Android privacy settings interface.

1. INTRODUCTION

Smartphones store a great deal of personal information, such as the user's contacts, location, and call history. Mobile operating systems use *permission systems* to control access to this data and prevent potentially malicious third-party applications ("apps") from obtaining sensitive user data. Part of the purpose of these permission systems is to inform and empower users to make appropriate decisions about which apps have access to which pieces of personal information.

The popular open-source Android mobile platform has used two general approaches to give users control over permissions. Initially, permissions were presented as an install-time ultimatum, or ask-on-install (AOI): at installation, an application would disclose the full list of sensitive resources it wished to access. Users could either grant access to all requested permissions or abort the installation entirely. Prior research has shown that most users do not pay attention to or do not these prompts when shown at install-time [12].

Recently, an *ask-on-first-use* (AOFU) permission system replaced install-time disclosures on Android. Under AOFU, the user is prompted when an application requests a sensitive permission for the first time. The user's response to this permission request carries forward to all future requests by that *application* for that *permission*. The AOFU approach, however, fails to consider that the user's preferences may change in different contexts. It only learns the user's preferences once under a certain set of contextual circumstances: the first time an application tries to access a particular data type. This system does not account for the fact that subsequent requests may occur under different contextual circumstances and therefore may be deemed less appropriate. For instance, a user might feel comfortable with an application requesting location data to deliver desirable location-based functionality. The same user, however, might find it unacceptable for the same application to access location for the purposes of behavioral advertising, possibly when the application is not even being used.

The *contextual integrity* framework can explain why AOFU is insufficient: it fails to protect user privacy because it does not account for the context surrounding data flows [25]. That is, privacy violations occur when a data flow (e.g., an app's access to a sensitive resource) defies user expectations. In recent work [38, 39], we showed that mobile users *do* make contextual privacy decisions: decisions to allow or deny access are based on what they were doing on their mobile devices at the time that data was requested.

In theory, asking the user to make a decision for every request is optimal, as the user will be able to account for the surrounding context and can then make decisions on a case-by-case basis. In practice, however, this results in unusable privacy controls, as the frequency of these requests could overwhelm the user [38]. Consequently, automating these decisions with machine learning yields a balance between

Copyright is held by the author/owner. Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee.
Symposium on Usable Privacy and Security (SOUPS) 2017, July 12–14, 2017, Santa Clara, California.

APROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

accurately implementing users' privacy preferences and not overburdening them with too many decisions [39]. Such automation requires the platform to have feedback mechanisms so that automated decisions can be reviewed and errors can be corrected, thereby yielding fewer future errors.

To this end, we designed a novel permission manager, TurtleGuard, which helps users to vary their privacy preferences based on a few selected contextual circumstances. It also provides information about the apps that they use, by providing a feedback loop for them to audit and modify how automated decisions are made. TurtleGuard allows users to (i) vary their decisions based on the visibility of the requesting application – our previous work showed that the visibility of the requesting application is a critical factor used by users when making mobile app privacy decisions [38], and (ii) have an improved understanding of how third-party applications access resources in the real world and under varying contextual circumstances.

We conducted an initial 400-person experiment to evaluate our preliminary design. Based on our analysis of this data, we then iterated on our design, conducting a 580-person validation study to demonstrate our design's effectiveness. Both experiments had four tasks: three tasks that involved using the system to locate information about current application permissions, and one task that involved modifying settings. We observed that participants who used TurtleGuard were significantly more likely to vary their privacy preferences based on surrounding circumstances than the control group. We believe that these results are a critical contribution towards empowering mobile users to make privacy decisions on mobile phone platforms. Our contributions are as follows:

- We present the first contextually-aware permission manager for third-party applications in Android.
- We show that when using our new interface (compared to the existing Android interface) participants were *significantly* more likely to both understand when applications had foreground versus background access to sensitive data and how to correctly control it.
- We show that our proposed interface has a minimal learning curve. Participants, who had never used TurtleGuard before, were as successful at accomplishing information retrieval tasks as those who used the existing Android interface.

2. RELATED WORK

The Android OS has thus far used two different permission models: ask-on-install (AOI) permissions, and ask-on-first-use (AOFU) permissions. Versions of Android before version 6.0 (Marshmallow) implemented ask-on-install permissions. Under this model, applications request that the user grant all permissions to the application at install time. The user must consent to all requested permissions in order to complete installation. Otherwise, if the user wishes to deny any permission, the only option available is to abort the installation entirely. Research has shown that few users read install time permissions, and fewer still correctly understand their meaning [12, 18].

Versions of Android from 6.0 (Marshmallow) onward use the AOFU permission model instead. Under AOFU, applications prompt users for sensitive permissions at runtime. These prompts protect access to a set of 24 "dangerous permissions," including geolocation data, contact lists, and SMS. Prompts appear when the application attempts to request protected resources for the first time. This has the advantage of giving users contextual clues about why an application requires a protected resource: users can consider what they are doing when the prompt appears to help determine whether to approve the request. Although AOFU offers an improvement over the install-time model in this regard, first-use prompts insufficiently capture a user's privacy preferences [39]. That is, the AOFU model does not consider scenarios where an application requests access to data under varying contexts.

Research on permission models has found that users are often unaware how apps access protected resources and how access may be regulated [12, 8, 11, 36, 34]. Shih et al. showed that users are more likely to disclose privacy information when the purpose is unclear [35]. Prior work has specifically analyzed location data: Benisch et al. show that a vast number of factors (time, day, location) contribute to disclosure preferences [5]; Reilly et al. show that users want minimal interaction with their technology [31]. Additionally, Patil et al. takes into consideration context: they suggest making feedback actionable and allowing for selective control regarding location data [29]. They also show that users have difficulty articulating location access controls, and suggest an interface that includes contextual factors as a potential solution [28]. Almuhimedi et al. studied AppOps, a permission manager introduced in Android 4.3 but removed in Version 4.4.2 [1]. AppOps allowed users to review and modify application permissions once installed, as well as set default permissions that newly installed applications must follow. They examined privacy nudges that were designed to increase user awareness of privacy risks and facilitate the use of AppOps. They concluded that Android users benefit from the use of a permission manager, and that privacy nudges are an effective method of increasing user awareness [1].

Although AppOps was removed from Android, Android 6.0 (*Marshmallow*) reintroduced permission management. It—and subsequent versions as of this writing—include an updated interface that allows the user to view all of the permissions that a particular app has been granted, as well as all of the apps that have been granted a particular permission (Figure 1). Unfortunately, these controls are buried deep within the Settings app, and it is therefore unlikely that users are aware of them. For instance, viewing a particular app's permissions requires navigating four levels of sub-panels, whereas viewing all the apps that have requested a particular permission requires navigating five levels. By comparison, TurtleGuard is one click from the main Settings panel and explicitly presents the relationships between ap- plications, permissions, and controls.

XPrivacy [6], DonkeyGuard [7], Permission Master [23], and LineageOS's[1] Privacy Guard [24] are examples of other third-party permission management software. These utilities require additional privileges and techniques to install because Android provides no official mechanisms for third-party programs to modify the permission system. For instance, Privacy Guard is built into the LineageOS custom ROM [24];

[1]LineageOS is a recent fork of CyanogenMod after the latter's discontinuation.

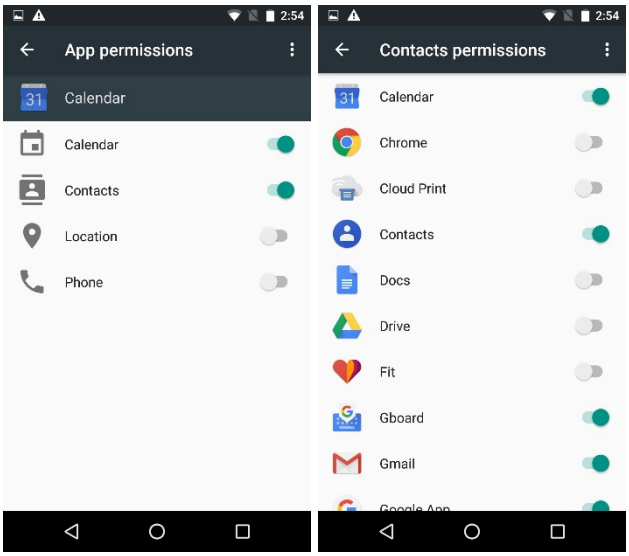


Figure 1: After navigating four and five levels of sub-panels within the Android Settings app, respectively, users can limit a specific app's access to specific permissions (left) or limit the apps that can access a particular permission (right).

others use the Xposed Framework [32], which requires an unlocked bootloader and a custom recovery partition. Such restrictions are necessary to prevent malicious software from interfering with the existing permission system.

Third-party permission managers offer users a variety of features to fine-tune access to sensitive resources on their devices. XPrivacy has the option to pass fake data to applications that have been denied access to protected resources [2]. Hornyack et al.'s AppFence similarly allows users to deny permissions to applications by providing fake data [16]. Providing fake data is more desirable than simply failing to provide any data at all, as the latter may cause functionality loss or application failures.

These managers follow an Identity Based Access Control model (IBAC), where individual permissions can be set for each app. Although this model allows users to specify fine-grained permission preferences, this may be ineffective in practice for two reasons. First, users may be overwhelmed by the number of settings available to them, some of which are only tangentially relevant to privacy. This security de- sign failure is known as the *wall of checkboxes* [14]. XPrivacy and Permission Master show controls for resources whose direct effects on user privacy are unclear, such as keeping a device awake. TurtleGuard improves usability by showing only controls for resources deemed "dangerous" in the Android platform [15] and others that previous research has shown are conducive to using run-time prompts [10]. Second, none of the existing permission managers display the context in which protected resources were accessed. XPrivacy, Donkey Guard, and LineageOS's Privacy Guard pro- vide timestamps for resource accesses, but the user does not receive important information about the app's state, such as whether it was actively being used when it requested access to sensitive data. Permission Master offers no historical information at all. TurtleGuard partially addresses this problem by listing recently allowed and denied permission access requests, along with the state and visibility of the requesting application at the time the permission was requested.

Apple's iOS platform offers visibility-sensitive location privacy settings: "Never" and "Always" (the two settings analogous to Android's permission on/o↵ toggles), and a "While using the app" option, which only permits an application to access location data while the application is active on the screen. TurtleGuard uses the same options, but our design is novel in both the extent of these settings and in who controls them. Apple's iOS allows developers to control which of the three options are available to users to select [3]. Application developers have faced criticism for removing the "While using the app" option, forcing users to choose between reduced functionality and granting the application unrestricted access to sensitive location data [26]. Our design, by contrast, gives users all three of these options for all *sensitive* permissions (Table 5, Appendix). Furthermore, developers cannot restrict user choice with these settings, as TurtleGuard is implemented in the operating system.

Wijesekera et al. show that although AOFU improves on install-time permissions, AOFU is insufficient because it does not account for the context of the requests [39]. They examined this by instrumenting the Android platform to log all instances of apps accessing sensitive resources. In addition to their instrumentation, the platform randomly prompted users about the appropriateness of various permission re- quests as those requests occurred. Participant responses to these prompts were treated as the dependent variable for a training set. Their study showed that 95% of participants would have chosen to block at least one access request had the system notified them. On average, participants would have preferred to block 60% of permission requests. Indeed, other work suggests that contextual cues are key in detecting privacy violations [25, 4].

A natural extension of AOFU is "ask on *every* use": rather than extrapolating the user's first-time preference to all future accesses to a given resource, each access instead requires user input. Such a model would conceivably allow users to accurately specify their contextual preferences because they know exactly which app attempted to gain access to what resource under which circumstance. This approach, however, is unusable in practice. Research has shown that applications request access to permission-protected resources with great frequency: on an average smartphone, roughly once every 15 seconds [38]. Such a high frequency not only risks habituation, but would render the device inoperable.

Recent research on permission models has turned towards using machine learning (ML) [39, 20, 21, 19]. One advantage is ML's ability to incorporate nuanced contextual data to predict user preferences; the approach has shown significantly lower error rates over the *status quo*, i.e., AOFU. Wijesekera et al. [39] also showed that ML reduces user involvement, thereby minimizing habituation. They emphasize, however, the importance of having a user interface that functions as a feedback-loop to the classifier, since no practical classifier will ever be 100% accurate. Users can use the interface to audit the decisions made by the classifier and correct any decisions that do not match their preferences.

Such a mechanism not only ensures that the classifier improves its accuracy over time, it also keeps users aware of decisions that were made on their behalves and informs them of how third-party apps are accessing sensitive resources under various circumstances.

TurtleGuard provides two core components necessary for usability under such contextual privacy models: we provide users with key contextual information when regulating access to sensitive resources, and we provide a method for users to audit and correct the decisions that have been automatically made by the system.

3. DESIGN OBJECTIVES

TurtleGuard's primary function is to inform users about the decisions that have been automatically made on their behalf, while allowing them to easily correct errors (thereby improving the accuracy of future decisions). These errors can be either false positives—an app is denied a permission that it actually needs to function—or false negatives—an app is granted access to data against the user's preferences.

Thompson et al. showed how attribution mechanisms can help users better understand smartphone application resource accesses [37]. They found that users expect this information to be found in the device's *Settings* app. In our initial experiment, we evaluated TurtleGuard as a standalone app, though for this reason we ultimately moved it within the Android *Settings* panel prior to our validation experiment.

3.1 Incorporating Context

In prior work, researchers observed that only 22% of participants understood that applications continue to run when not visible and that they have the same access to sensitive user data that they do when actively being used [37]. This means that the majority of users incorrectly believe that applications either stop running when in the background or lose the ability to access sensitive data altogether. Wijesekera et al. corroborated this observation in a field study of users' privacy expectations: users are more likely to deem permission requests from background applications as being inappropriate or unexpected, and indicate a desire to regulate applications' access to sensitive data based on whether or not those applications are in use [38].

In the default permission manager, users cannot vary their decisions based on the visibility of the requesting application, or any other contextual factors. Our overarching goal is to empower users to make contextual decisions (i.e., based on what they were doing on the device) and to apply these decisions to future use cases, so that fewer decisions need to be explicitly made overall. As a first step towards allowing users to make contextual decisions, TurtleGuard makes decisions about whether or not to allow or deny access based on whether the requesting application is actively being used. While this is but one contextual factor amongst many, it is likely one of the most important factors [38].

Moving one step beyond the all-or-nothing approach of allowing or denying an application's access to a particular data type, our new design gives the user a third option: allowing applications to access protected data only *when in use* (Table 1 and Figure 2). When the *when in use* option is selected, the platform only allows an application to access a resource if the application is running in such a way that it is *conspicuous* to the user of the device. We consider the following behaviors conspicuous: (i) the application is running in the foreground (i.e., the user is actively using it), (ii) the application has a notification on the screen, (iii) the application is in the background but is producing audio while the device is unmuted. If these conditions do not hold, then access to the resource is denied.

option	meaning
always	The permission is always granted to the requesting application, regardless of whether the application is running in the foreground or background.
when in use	The permission is granted to the requesting application only when there are cues that the application is running, and denied when the application is running invisibly in the background.
never	The permission is never granted to the requesting application.

Table 1: The three possible permission settings under TurtleGuard. The *when in use* option accounts for the visibility of the requesting app, which is a strong contextual cue.

3.2 Auditing Automatic Decisions

Although Android currently provides an interface to list the applications that recently accessed location data, similar information is unavailable for other protected resources. The existing Android interface also does not differentiate between actions that applications take when *in use* and when *not in use*. TurtleGuard's main design objective is therefore to communicate the types of sensitive data that have been accessed by applications and under what circumstances.

Our initial design of TurtleGuard can be seen in Figure 2. The first tab (activity) shows all of the recently allowed or denied permission requests, including when those requests occurred and whether the application was in use at the time. TurtleGuard presents this information as a running timeline—a log sorted chronologically. A second tab lists all of the apps installed on the phone in alphabetical order, allowing the user to examine what decisions have been made for all permissions requested by a particular app. The user can expand a log entry to change future behavior, if the platform's automated decision to allow or deny a permission did not align with the user's preferences. When the user uses this interface to change a setting, the classifier is retrained based on the updated information.

3.3 Permission Families

Android uses over 100 permissions and a given resource can have more than one related permission. Felt et al. found that not all the permission types warrant a runtime prompt—it depends on the nature of the resource and the severity of the threat [9]. Consequently, TurtleGuard only manages a subset of permissions (Table 5, Appendix) based on those deemed sensitive by prior work and by the latest Android version. In the first prototype of TurtleGuard, we had listed the original names of the permissions, ungrouped. One of the changes we made as we iterated on our design after our pilot experiment was to implement permission "families". For

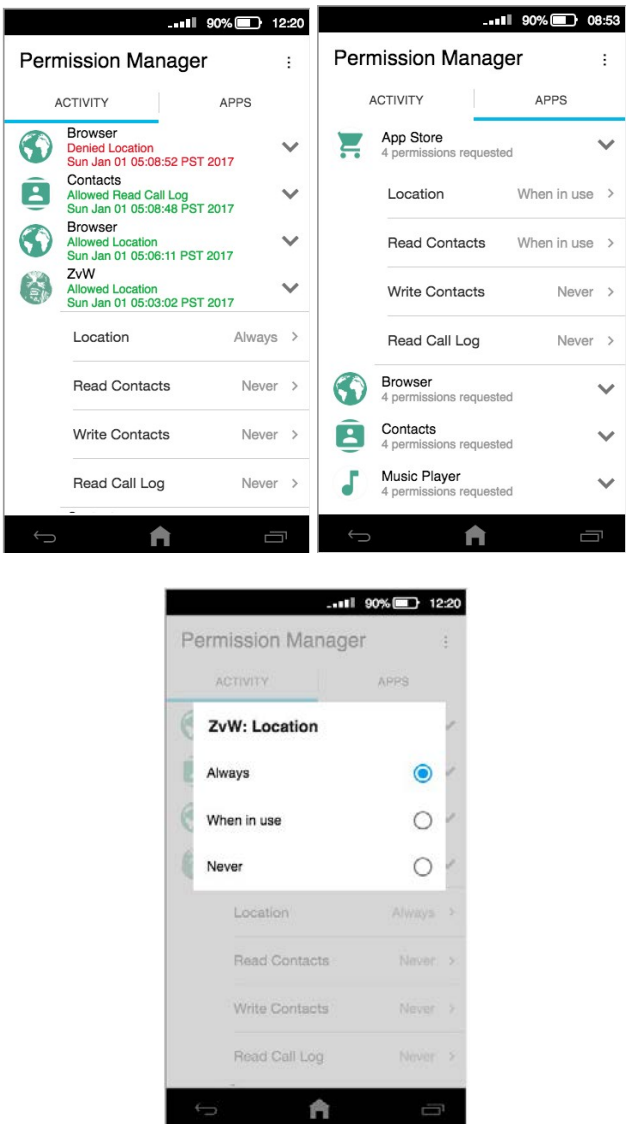


Figure 2: The pilot design of TurtleGuard listed recent app activity (top left), a list of installed apps and their associated permissions (top right). Permissions can be *always* granted, granted only *when in use*, or *never* granted (bottom).

example, read contacts and write contacts are grouped into a single contacts permission family. This means that within TurtleGuard, users only see the human- readable resource type and not the underlying permissions the family manages. Any changes that a user makes about granting a resource therefore affects all permissions in the same family. For example, there is no longer a distinction between coarse and fine location data; both are either allowed or denied by a location settings change made using the TurtleGuardinterface.

4. METHODOLOGY

We conducted two online experiments to evaluate the effectiveness of TurtleGuard at providing users with information and control over app permissions, as compared to Android's default permission manager (as of versions 6.0). We designed the first experiment to examine our initial prototype, as described in the previous section. Based on the analysis of our first experiment, we made changes to our design, and then validated those changes through a second experiment. In both experiments, we asked participants to perform four different tasks using an interactive Android simulation. These tasks involved either retrieving information about an appli- cation's prior access to sensitive resources or preventing ac- cess in the future (i.e., modifying settings). Our study was approved by our IRB (#2013-02-4992).

In both experiments, we randomly assigned participants to either the *control* or *experimental* conditions. We presented *control* participants with an interactive HTML5 simulation of the default permission manager, which is accessible from within the Settings app. We presented *experimental* participants with an interactive HTML5 simulation of our novel permission manager, TurtleGuard. During our pilot experiment, TurtleGuard was accessible through an icon on the home screen labeled "Privacy Manager," though we added it as a sub-panel to the Settings app prior to the validation experiment (Figure 6 in the Appendix). The questions and tasks for participants were identical for the two conditions and both experiments.

4.1 Tasks

We presented participants with four tasks to complete using the interactive Android simulations: three tasks to retrieve information about permission settings, and one task to modify permission settings. Some of these tasks required participants to find information about a specific app's abilities. In order to avoid biases from participants' prior experiences and knowledge of specific real-world apps, these questions instead focused on a fictitious app, *ZvW*. While we randomized the order of the tasks, we ensured that Task 3 always came before Task 4 (i.e., we never asked them to prevent background location data collection prior to asking them whether background location data was even possible). After each task, we asked participants to rate the difficulty of the task using a 5-point Likert scale ("very easy" to "very difficult"). Finally, upon completing all tasks, we asked them several demographic questions and then compensated them $2. We now describe the four tasks in detail.

Task 1: What were the two most recent applications that accessed this device's location?

In this task, we asked participants to use the Android simulation and identify the two applications that most-recently accessed location data. Participants used two open-ended fields to answer this question. In the *control* condition, this task was correctly accomplished by navigating to the "location" screen from within the Settings application (Figure 3). This screen presents information about applications that recently requested location data.

In the *experimental* condition, this task was accomplished by simply studying the "activity" screen, which was displayed immediately upon opening TurtleGuard (Figure 2). Given that this task was already supported by the default permission manager, we wanted to verify that TurtleGuard performed at least as well.

Task 2: Currently, which of the following data types can be accessed by the ZvW application?
In the *control* condition, this was accomplished by performing the four steps to access the screen in Figure 4 (right): selecting the "Apps" panel within the Settings app (Figure 3, left), selecting the ZvW application, and then selecting the "Permissions." This screen depicted a list of permissions available to the application based on what the application declares as its required permissions; the user is able to fine-tune this by selectively disabling certain permissions using the sliders on this screen. We wanted participants to identify the permissions that were enabled, rather than all of those that *could* be enabled in the future.

In the *experimental* condition, participants could accomplish this task by selecting the "Apps" tab from within TurtleGuard and then expanding the ZvW application to view its requested permissions (Figure 2, top right). In both conditions, the correct answer to the question was that "location" is the only data type that can be accessed by the ZvW application. Again, given that this task was already supported by the default permission manager, we wanted to verify that TurtleGuard performed at least as well.

Task 3: Is the ZvW application able to access location data when it is not being actively used?
We designed this task to determine whether TurtleGuard was effective at communicating to participants in the *experimental* condition the difference between foreground and background data access. Similarly, we wanted to examine whether participants in the *control* condition understood that once granted a permission, an application may access data while not in use. Based on the settings of the simulations, the correct answer across both conditions was "yes."

Participants in the *control* group must navigate to Settings, then the "Apps" panel, and view the list of permissions corresponding to the ZvW application, similar to Task 2. Location is turned on, and so participants must be able to understand that this means that the permission is granted even when it is not actively being used. Participants in the *experimental* condition can use TurtleGuard's "Apps" tab to view the requested permissions for the ZvW application. This shows that the location permission is listed as "always" (Figure 2, top right) and that "when in use" is an unselected option (Figure 2, bottom).

Task 4: Using the simulation, prevent ZvW from being able to access your location when you aren't actively using ZvW (i.e., it can still access location data when it is being used). Please describe the steps you took to accomplish this below, or explain whether you believe this is even possible.
As a follow-up to the third task, the fourth task involved participants explaining the steps that they went through in order to limit background location access, or to explain that it is not possible.

Those in the *experimental* condition could locate and change this permission setting either through the activity timeline or by locating ZvW from the "Apps" tab (Figure 2). We marked answers correct that specifically mentioned changing the setting to "when in use."

Those in the *control* condition could not prevent this access. We marked responses correct if they indicated that this task was impossible to complete. Two coders independently reviewed the responses to this task (Cohen's $\kappa = 0.903$). The objective of this task was to see TurtleGuard's success at allowing participants to vary settings based on application use (a strong contextual cue) and to examine whether participants knew that this was not possible when using the default permission manager.

4.2 UI Instrumentation
We built an interactive HTML5 simulation of the UI designs described in the previous section using *proto.io*. We instrumented the simulations to log all interactions (e.g., panels visited, buttons clicked, etc.). This data allowed us to analyze how participants navigated the UI to perform each task.

4.3 Qualitative Data
In addition to analyzing the participants' responses to the four tasks, their perceived difficulty of each of the tasks, and their demographic information, we also collected responses to two open-ended questions:

Thinking about the tasks that you performed in this survey, have you ever wanted to find similar information about the apps running on your smartphone?
We coded participants' responses as a binary value. Responses indicating sentiments such as "yes" and "I always wanted that" were coded as true. Clear negative answers and weak affirmative answers such as "sometimes" and "maybe" were coded as false. The purpose of this question is to see how prevalent seeking information is in the real world.

Thinking about the simulation that you just used, what could be done to make it easier to find information about how apps access sensitive information?
We coded participants' responses in multiple ways. First, as binary values indicating contentment with the presented design. Responses that affirmed that the user would change nothing about the presented design were coded as true. Any complaints or suggestions were coded as false, as well as responses with uncertainty, confusion, or ambivalence (e.g., "I don't know"). We further coded responses that had specific suggestions, using tags for the different themes.

Each response was coded by two experienced coders working independently, who then compared responses and recorded their coding conflicts. The coders discussed and reconciled the differences using their mutually agreed upon *stricter* interpretation given the nature of the tasks. This produced the final coding of the data, which is used in our analysis.

5. PILOT EXPERIMENT
Using the methodology outlined in the previous section, we recruited 400 participants from Amazon's Mechanical Turk for a pilot experiment. We discarded 8 incomplete sets of responses, leaving us with 392 participants. Our sample was biased towards male respondents (65% of 392), however, a chi-square test indicated no significant differences between genders with regard to successfully completing each task. Disclosed ages ranged from 19 to 69, with an average age of 33. In the remainder of this section, we describe our results for each task, and then describe changes we made to

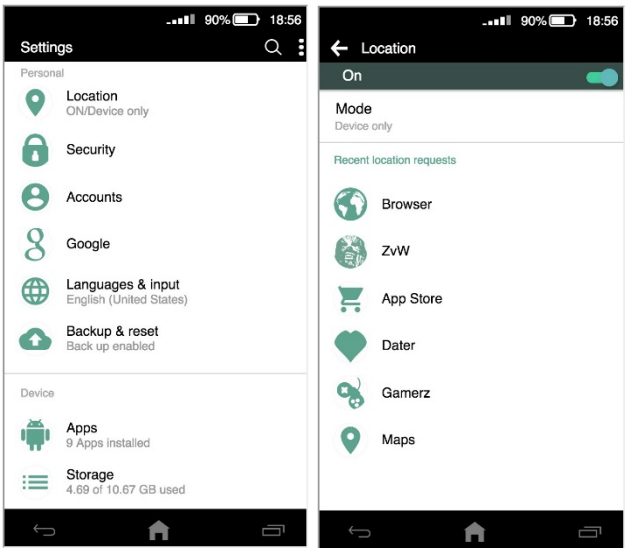


Figure 3: In Task 1, participants in the *control* condition could identify the most recent applications that requested location data from within the Settings application. This was also a valid method for Task 1 in the *experimental* condition for the validation study.

TurtleGuard's interface as a result of this initial experiment. We note that in our simulation, *Settings* can only be reached by tapping on the icon from the home screen. In all of our tasks, we also asked participants to evaluate perceived difficulty using a 5-point Likert scale.

5.1 Task 1: Recent Location Access

In the *control* condition, 84% of participants (167 out of 198) correctly completed this task, whereas only 68% (132 out of 194) completed it correctly in the *experimental* condition. This difference was statistically significant ($x^2 = 14.391$, $p < 0.0005$), though with a small-to-medium effect size ($c/ = 0.192$). In both cases, answers were marked correct if they mentioned both the Browser and ZvW applications (Table 2). Of the 49 participants in the *experimental* group who tried but failed, 13 never opened TurtleGuard, and over 73% (36 of 49) entered"Browser"and"Contacts", which were the first two applications listed in the activity tab of the Permission Manager. The activity tab showed recent resource accesses in a chronological order—"Browser" had been denied a *location* request and "Contact" had successfully accessed *calllogs*.

Participants did not seem to understand that the activity log presented entries related to *all* sensitive data types, not just location data. This confusion might also stem from their familiarity with the location access panel in stock Android, in which location access requests are presented in chronological order. We hypothesize that this confusion is addressable by redesigning the activity log to better distinguish between data types and allowed-versus-denied permission requests. One possible way of implementing this is to create separate tabs for allowed and denied requests, as well as to group similar data types together (rather than presenting all permission request activity in chronological order).

Condition	Correct	Incorrect
Task 1		
control	167 (84%)	31 (15%)
experimental	132 (68%)	62 (32%)
Task 2		
control	140 (70%)	58 (29%)
experimental	116 (59%)	78 (40%)
Task 3		
control	86 (43%)	112 (56%)
experimental	153 (78%)	41 (21%)
Task 4		
control	47 (23%)	151 (76%)
experimental	144 (75%)	49 (25%)

Table 2: Participants in each condition who performed each task correctly during the pilot experiment.

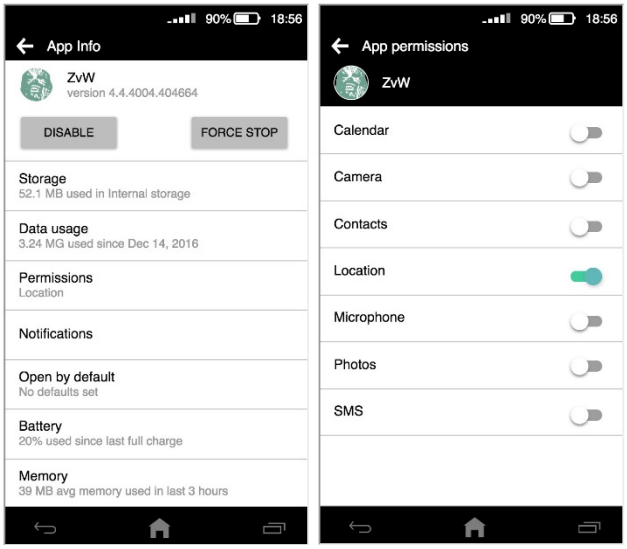


Figure 4: In Task 2, participants in the *control* condition could identify the permissions granted to the ZvW application by selecting the "Apps" panel from within the Settings application, and then selecting the application, followed by the "Permissions" panel.

5.2 Task 2: Finding Granted Permissions

In the second task, we asked participants to list all of the data types that the ZvW application *currently* had access to. We observed that 140 participants in the *control* condition (70.7% of 198) and 116 participants in the *experimental* condition (59.8% of 194) performed this task correctly. After correcting for multiple testing, this difference was not statistically significant ($x^2 = 5.151$, $p < 0.023$).
However, despite the lack of statistical significance, we were surprised that not more people in the *experimental* condition answered correctly. Upon investigating further, we noticed several confounding factors that might have made this task more difficult for people in this condition. First, while the *control* condition displays the currently-allowed permissions as grayed-out text on the "App Info" page (Figure 4), the *experimental* condition lists all *requested* permissions—

which is a superset of the allowed permissions (top-right of Figure 2). Second, we noticed that due to an experimental design error, the permissions requested by the ZvW app in the *experimental* condition included several that were not included in the options presented to participants (e.g., "Write Contacts" and "Read Call Log"). This may have made this task confusing for these participants.

5.3 Task 3: Finding Background Activity

In the third task, we asked participants whether the ZvW application had the ability to access location data while not actively being used. We observed that 86 participants in the *control* condition (43% of 198) correctly answered this question, as compared to 153 participants in the *experimental* condition (78% of 194). This difference was statistically significant ($x^2 = 51.695$, $p < 0.0005$) with a medium effect size ($c/ = 0.363$). Thus, the new dashboard interface successfully differentiated between foreground and background permission usage.

5.4 Task 4: Limiting Background Activity

We observed that only 47 participants in the control condition (23% of 198) correctly stated that this task was impossible. In the *experimental* condition, 144 (74% of 193)[2] clearly articulated the steps that they would go through using the privacy dashboard to change location access from "always" to "when in use." This difference was statistically significant ($x^2 = 101.234$, $p < 0.0005$) with a large effect size ($c/ = 0.509$).

5.5 Design Changes

Based on the results of our first two tasks, in which participants in the *control* condition were more likely to correctly locate information about recent app activities and the permissions that apps had requested, we made several design changes to the TurtleGuard interface. First, we split the activity timeline into two separate tabs: recently allowed permission requests, and recently denied permission requests. Second, rather than showing all activity in chronological order, the activity timeline is now categorized by resource type, with the events for each resource type sorted chronologically. These changes can be seen in the top of Figure 5.

In addition to these changes, we also modified the apps tab to show grayed-out allowed permissions for each app, similar to the App Info panel in the default permission manager. Due to the error we noted in the *experimental* condition in Task 2, we made sure that all app permissions were the same in both conditions.

Finally, we moved TurtleGuard to be within the Settings app, so that it appears as a panel labeled "Permissions Man- ager" (Figure 6, Appendix). For consistency, when participants in the *experimental* condition select the "Permissions" subpanel from within the "App Info" panel (Figure 4, left), they are now redirected to TurtleGuard's "Apps" panel, preopened to the app in question (Figure 5, bottom right).

6. VALIDATION EXPERIMENT

Following our pilot experiment and subsequent design changes, we performed a validation experiment. In the remainder of this section, we discuss our results (Table 3).

[2]One person could not load the `iframe` containing the simulation during this task.

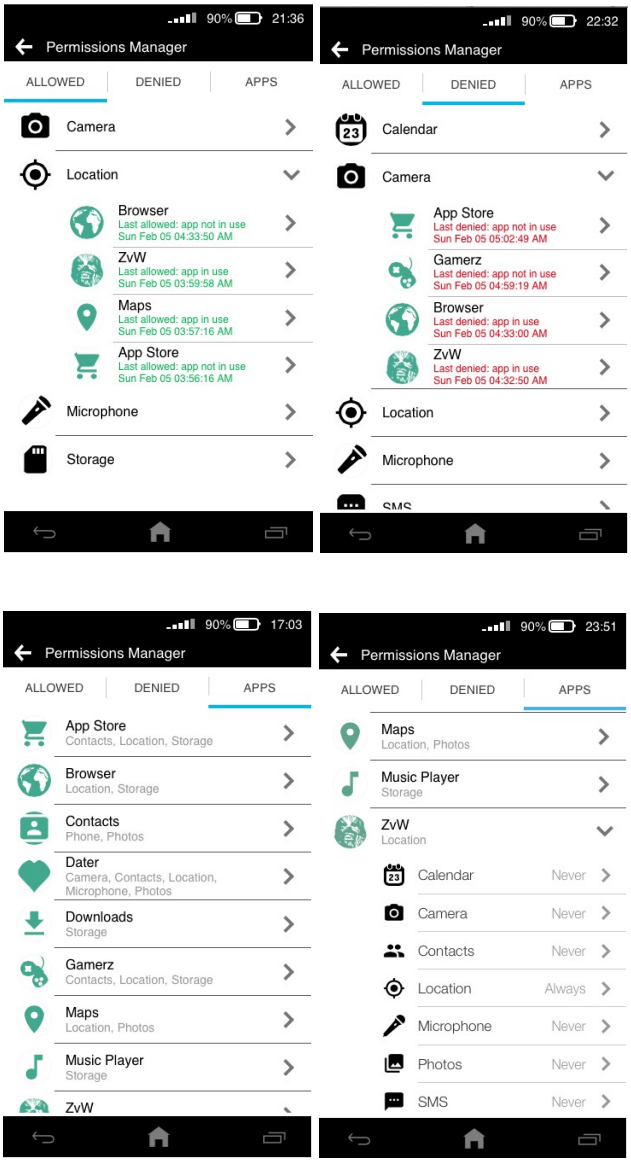


Figure 5: TurtleGuard separates recently allowed (top left) and denied (top right) permissions. The "Apps" tab lists the allowed permissions of all apps (bottom left). Expanding an app allows the user to make changes (bottom right).

6.1 Participants

Because of several known biases in Mechanical Turk's demographics [27, 33, 22], we decided to compare a sample of 298 Mechanical Turk participants to a sample of 300 Prolific Academic participants. Peer et al. recently performed several studies on various crowdsourcing platforms and concluded that the latter yields more diverse participants [30]. We limited both groups to participants based in the U.S., over 18, owning an Android phone, and having a 95% approval rating on their respective platform. After removing 18 incomplete responses, we were left with a combined sample of 580 participants. We analyzed the results from the two groups, and discovered that the high-level findings (i.e.,

Condition	Correct	Incorrect
Task 1		
control	237 (82.6%)	50 (17.4%)
experimental	241 (82.5%)	52 (17.5%)
Task 2		
control	232 (77.1%)	55 (22.9%)
experimental	226 (80.8%)	67 (19.2%)
Task 3		
control	108 (37.6%)	179 (62.4%)
experimental	230 (78.5%)	63 (21.5%)
Task 4		
control	79 (27.5%)	208 (72.5%)
experimental	224 (76.5%)	69 (23.5%)

Table 3: Participants in each condition who performed each task correctly during the validation experiment.

task performance) did not observably differ. For the remainder of our study, we therefore discuss the combined results. Our sample was biased towards male respondents (63% of 580), however, a chi-square test indicated no significant differences between genders with regard to successfully completing each task. Disclosed ages ranged from 19 to 74, with an average age of 33. Participants performed the same tasks as those in the pilot experiment and took on average 9 minutes and 17 seconds to complete the experiment.

6.2 Task 1: Recent Location Access

Recall that in this task, we asked participants to identify the two most recent applications that accessed location data. For the *experimental* condition, in addition to using the same method as the *control* (navigating to the "Location" sub-panel of the Settings app), participants could navigate to the "Allowed" tab within TurtleGuard, and then examine the"Location"permission to see the two most recent accesses (top left of Figure 5). In the *control* condition, 237 participants (82.6% of 287) correctly completed this task, whereas 241 (82.5% of 293) completed it correctly in the *experimental* condition. A chi-square test revealed that this difference was not statistically significant ($x^2 = 0.011$, $p < 0.918$).

We observed that most of the participants in both conditions used the default method of accomplishing this task (i.e., accessing the Location sub-panel): 80.1% of those who answered correctly in the *experimental* condition and 92.8% of those in the *control* condition. Fifteen participants in the *control* condition answered correctly despite not accessing the panel—likely by random guessing, and two who answered correctly by exhaustively searching the "App Info" panels of installed apps, to see which had been granted the location permission; 48 participants in the *experimental* condition used TurtleGuard to yield the correct answer.

A total of 102 participants incorrectly answered the question in Task 1. Of the incorrect responses, five participants failed to properly navigate the simulation and wrote that it was broken or the buttons did not work; 9 participants did not respond or wrote that they did not know. Of the other 88 participants, 38 (43%) listed "App Store" as one of their selections, making it the most common error.

More specifically, 18 participants listed their answers as both "App Store" and "Browser." We believe that this is because both the stock Android Apps Manager and TurtleGuard's "Apps" tab (Figure 5, bottom) sort the entries alphabetically, and by looking at the permissions available to both of these apps, participants would see that both have location access. Nevertheless, they are not the most *recent* apps to access location data.

Overall, these results suggest that the changes we made after our pilot experiment resulted in marked improvements. We further investigated this by examining participants' perceived ease-of-use, as measured using the 5-point Likert scale ("very easy (1)" to "very difficult (5)"). In the *experimental* condition, 84 participants accessed TurtleGuard to complete this task (regardless of whether or not they answered correctly). We compared these 84 responses with the 463 responses from participants who only used the default Settings panel (i.e., 195 in the *experimental* group and 268 in the *control* group). The median responses from both groups was "easy" (2), however there was a statistically significant difference between the groups (Wilcoxon Rank-Sum test: $Z = 3.9605$, $p < 0.0005$), with a small effect size ($r = 0.17$)—participants who used TurtleGuard found it more difficult compared to the default Settings panel. This difference appears to be due to those who performed the task incorrectly: the median response for TurtleGuard users who answered incorrectly was "difficult (4)," whereas it was "neutral (3)" for other participants. This may actually be a good thing: participants who confidently answered incorrectly are at greater risk due to over confidence, whereas those who had difficulty may be more likely to seek out more information.

6.3 Task 2: Finding Granted Permissions

In this task, participants had to locate the app's allowed permissions to discover that "location" was the only allowed permission in both the *experimental* and *control* conditions. This could be accomplished by viewing TurtleGuard's Apps tab (bottom of Figure 5) for those in the *experimental* condition, or by viewing an app's App Info panel from within the Settings app (Figure 4), which was available to those in either condition.

In total, 458 participants correctly performed this task (79% of 580). Table 3 displays the breakdown of the results by condition. A chi-square test did not yield statistically significant results between the two conditions in terms of task completion ($x^2 = 0.984$, $p < 0.321$).

Of the 226 *experimental* condition participants who performed the task correctly, 127 (56.2%) did so by using TurtleGuard. In total, 145 *experimental* condition participants accessed TurtleGuard, and reported a median task difficulty of "easy (2)." This did not significantly differ from the 375 other participants in both conditions who only examined the default Settings panels to perform the task and also reported a median difficulty of "easy" ($Z = 1.808$, $p < 0.238$); 60 participants never opened Settings (10 of whom answered the question correctly, likely due to random guessing).

6.4 Task 3: Finding Background Activity

To perform this task, participants in the *control* group had to navigate to Settings, then the "Apps" panel, and view the list of permissions corresponding to the ZvW application

(Figure 4). However, performing this sequence of steps still did not guarantee they would answer the question correctly: they needed to observe that location data was allowed, as well as understand that this meant that location data could be accessed by the app even when it is not actively being used. Participants in the experimental condition answered this question through TurtleGuard, which shows that the location permission was listed as "Always" (Figure 5), thereby eliminating the ambiguity.

We observed that 230 *experimental* condition participants answered this question correctly (78.5% of 293), as compared to only 108 *control* participants (37.6% of 287). A chi-square test showed that this difference was significant ($x = 97.914$, $p < 0.0005$) with a medium-to-large effect size ($c/ = 0.414$). This observation corroborates Thompson et al.'s findings [37] that users are largely unaware that apps can access sensitive data when not in use. TurtleGuard, however, was more effective at communicating this information to participants. Among the participants in the *experimental* condition, 24.57% took the extra step to click on the location entry (bottom right of Figure 5) to see the other options available (Figure 2): *always*, *when in use*, and *never*.

We found that 129 participants used TurtleGuard to per- form this task, which suggests that 101 (34.5% of *experimental* condition participants) still got it correct either based on prior knowledge—a proportion consistent with Thompson et al.'s findings [37]—or after having used TurtleGuard in preceding tasks. There were 383 participants who completed the task by examining existing areas of the Settings app, whereas 68 participants never bothered to open Settings to complete this task. The median ease of use for those who used TurtleGuard was "easy (2)", while the median ease of use for those who used the default permission manager was "neutral (3)". This difference was statistically significant ($Z = 2.885$, $p < 0.004$) with a small effect size ($r = 0.13$). Participants in the *control* condition also took significantly longer to complete the task: 49.63 seconds versus 26.65 seconds. A Wilcoxon Rank-Sum test found this difference to be statistically significant ($Z = -5.239$, $p < 0.0005$, $r = 0.22$).

6.5 Task 4: Limiting Background Activity

Task 4 asked participants to describe the steps to prevent an application from accessing location data while the application was not in use, or to state that it is not possible to prevent it. It is only possible to prevent it using TurtleGuard.

In the *experimental* condition, 224 (76.5% of 293) explic- itly stated how they would use TurtleGuard to change the permission to "when in use",[3] whereas only 79 (27.5% of 287) *control* group participants correctly stated that this task was impossible using the default permission manager. This difference was statistically significant ($x^2 = 137.14$, $p < 0.0005$) with a large effect size ($c/ = 0.49$).

A majority of the participants (72.5%) in the *control* group incorrectly believed that they could vary their decisions based on the *visibility* of the application. The most common responses involved disabling location data altogether, preventing the app from running, or restricting "background data":

- Settings > Apps > ZvW > Toggle Location O↵
- Disable or Force Stop the Application
- Settings > Location > ZvW > Permissions > Toggle Location O↵
- Settings > Apps > ZvW > Data Usage > Restrict Background Data
- Settings > Location > Toggle Location O↵

A considerable portion (14%) chose to "restrict background data," which does something else entirely: it prevents data surcharges while roaming on foreign networks. This is another example of a disconnect between users' mental models and the true meaning of these configuration options. That said, a small number of participants in the *control* condition correctly stated that they would need to disable the app's location permission, and then re-enable it every time they wanted to use that app, a tedious process that is prone to forgetfulness—we treated this response as correct. Another substantial portion among the default permission manager condition (46%) wanted to block the location globally (from the default location panel) or block the location access from ZvW app entirely. While this is an overly restrictive option compared to *when in use*, this is the closest option provided in Android—we treated this as an incorrect response.

As expected, participants in the *control* condition rated the difficulty of this task as "neutral (3)", whereas the median Likert score from those in the *experimental* condition was "easy (2)". This difference was statistically significant with a large effect size ($p < 0.0005$, $c/ = 0.49$). The participants in the *control* condition who successfully completed the task (e.g., by acknowledging it was impossible) struggled immensely with it, evaluating it as "difficult (4)".

7. USER PERCEPTIONS

After completing the four tasks, participants answered two open-ended questions about whether they have looked for this type of permission information in the past, and whether they have any suggestions to offer us about the design of the interface they had just used. Two researchers independently coded each question and then resolved conflicts. We provide Cohen's inter-rater reliability statistic ( ) for each coding.

7.1 Prior Experiences

Our first question asked: *Thinking about the tasks that you performed in this survey, have you ever wanted to find similar information about the apps running on your smartphone?*

Our goal was to determine whether participants had previously thought about resource access or configuring privacy preferences, and whether having these features would be beneficial. On average, 63.1% of participants stated that they had thought about this (Cohen's $\kappa = 0.792$), and the experimental condition they were in proved to be insignificant. We did, however, observe a positive correlation between performance on the four tasks (i.e., number of tasks performed correctly) and reporting having previously thought about these issues ($p < 0.007511$, $r = 0.155$).

Among the people who chose to be more detailed in their responses, several themes emerged. A large portion mentioned that the reason they had tried these tasks before is that they wanted to be able to exert more control over their installed apps:

[3]We used a very conservative rubric: 11 participants who described using TurtleGuard, but did not explicitly use the phrase "when in use," were coded as being incorrect.

	Changes	No Changes
control	245 (85.4%)	42 (14.6%)
experimental	187 (63.8%)	106 (36.3%)

Table 4: Whether participants believed changes were needed to the interfaces they used during the validation study.

- "*I was somewhat familiar with these menus already before starting this task. I like to have control over my app permissions including location and data management.*"
- "*Yes, I've often wanted a little more control over what my apps get to access*"

A minority of participants expressed their frustrations on how the current default user interfaces in Android were confusing and did not let them set privacy preferences the way they wanted:

- "*Yes but usually can't find anything on there either like these. So I gave up trying.*"
- "*Yes. I want to know what they collect, although it gets tedious to try to figure it all out. Sometimes I'd rather just ignore it.*"

These comments highlight the fact that many users want to have control over resource usage by applications, and that many feel helpless to do so, given the options offered by current privacy management interfaces. These observations further strengthen the need for a more usable interface that will help people to feel more empowered.

7.2 Suggestions

In our second exit survey question, we asked: *Thinking about the simulation that you just used, what could be done to make it easier to find information about how apps access sensitive information?*

This question had two purposes: (i) to gather specific design recommendations from participants who used TurtleGuard; (ii) to get general suggestions from participants who used the default permission manager.

In total, 66.03% participants (383 of 580) suggested at least some change or improvement (Cohen's κ = 0.896). Table 4 shows the breakdown of how many participants in each condition prefer a change in the dashboard within their condition. A chi-square test shows a statistically significant association between a participant's condition and whether the participant wants changes in the dashboard ($p < 0.00005$, $c/$ = 0.237). This suggests the participants in the *experimental* condition are more satisfied with the controls provided by the new design than those in the *control* condition. Our work aims to fill the need users have regarding control over permissions and their personal privacy.

The most common suggestion (32.24% of all suggestions) was to reduce the number of layers to the actual permission interface (Cohen's κ = 0.603). Participants complained about number of different interfaces they had to traverse before reaching the actual permission interface. Many participants suggested that they would prefer to reach a permission control interface directly through the application—either as part of the application or by pressing the app icon. TurtleGuard addresses this concern by providing a path to permission management that involves fewer clicks and centralizes all permission management functionality.

- "*Streamline the interface to require less touches to find the information about permissions and make it explicit as to what type of data would be collected if allowed.*"
- "*Perhaps have an easier way to access the app's settings, such as holding onto an app's icon will bring up its specific settings.*"
- "*Make each app itself have the option to find that information instead of going to the general phone settings.*"
- "*There should be one centralized location, or maybe an app for that. Just to toggle with these very important settings.*"

Seven participants thought having a log of recent resource usage by applications would be useful. Some went further, mentioning that the log should also provide contextual cues, such as the visibility of the application at the time it makes the request. This finding provides evidence in support of Liu et al. [20], that recent statistics help users make better decisions. TurtleGuard provides this functionality by showing all the recent resource requests along with (i) the decision that platform took on behalf of the users, (ii) the time that the decision was made, and (iii) the visibility of the requesting application.

- "*It would be useful to have a dashboard which shows which apps are accessing what and when. Being able to see a log of the actual data that was accessed would also be useful.*"
- "*A log could be provided as an option in the settings that shows all times an app accessed sensitive information.*"

A few participants (14.6%) also suggested that there should be a tutorial, wizard style guide, or a FAQ to explain how to manage permissions (Cohen's κ = 0.651). Some wanted the applications to explain *why* they need access to certain resources. Some even suggested runtime prompts for every sensitive request access. One participant suggested that app developers hold a YouTube Q&A session on resource usage after each release:

- "*As the app is being introduced to the users, they should make a youtube q&a to answer any simple questions like this.*"

Prior work has already shown that having runtime prompts on every sensitive request is not feasible [38]—we believe that a log of recent resource accesses with surrounding context is the closest practical solution.

8. DISCUSSION

Our primary goal is to empower users to make privacy decisions better aligned with their preferences and to keep them informed about how third-party applications exercise granted permissions, and under what circumstances. We

performed iterative user-centered design on a new permission management interface, TurtleGuard, which offers users significant improvements in their ability to control permissions when compared to the default permission manager.

8.1 Auditing Automated Decision Making

Recent research uses machine-learning techniques to automatically predict users' permission preferences [39, 20, 19, 21]. While machine-learning (ML) techniques have been shown to be better at predicting users' preferences [39], they are still prone to errors.

If systems are going to use ML in the future, there must be mechanisms for users to audit the decisions made on their behalves. We believe that the design we present in our study is a critical first step towards achieving that goal. Participants using TurtleGuard were better able to understand and control *when* apps have access to sensitive data than participants using the default permission manager. A substantial proportion of participants mentioned the desire to have a log that they could use to see how each application accesses sensitive resources—functionality that is missing in the default permission manager, but is provided by TurtleGuard.

8.2 Correcting Mental Models

In Task 4, we asked participants to disable access to location data when the example app, ZvW, was not actively being used, or to explain that this was not possible. We found that 72.5% of the participants in the *control* condition incorrectly believed that this was possible. Analyzing the different paths that participants in the *control* condition took while using the Android simulation, it was evident that the majority of participants did not understand the limits of the permission interface's provided functionality. This mismatch between users' mental models and actual functionality may lead to users incorrectly believing that they have denied access to certain requests for sensitive data.

8.3 Privacy Nudges

Previous work investigated ways to nudge users to configure their privacy settings and make them aware of how applications access their data [20, 13, 17]. While helping motivate users to use TurtleGuard (and other privacy management interfaces) is important, it is out of scope for this work. Nevertheless, our survey results showed that 63.1% of participants—independent of condition—previously searched for permission information on their smartphones. This shows that users are keen to understand how applications use their sensitive resources, and interfaces similar to the one we present in this study fill a critical need.

8.4 Limitations

In our proposed interface, TurtleGuard, we allow users to vary their decisions based on the visibility of the requesting application. We believe this is a significant first step towards enabling users to make contextual privacy decisions. The full extent of the impact of the surrounding context, however, goes beyond the mere visibility of the requesting application. More work is needed to understand different contextual factors and their respective impact on users' privacy decisions. We hope the results of this study will pave the path for future work on implementing *fully* contextually aware permission managers.

Additionally, we acknowledge the limitations in our screening process: participants who selected Android as their mobile device may have varying levels of usage and knowledge regarding the platform. Prior experience may have rendered the default permission manager as being easier to use for some participants in the *control* condition. This suggests that for new Android users, the usability improvements of TurtleGuard may be even greater than what we observed.

We also acknowledge that irregularities in the simulation may have had an impact towards participants' comprehension and completion rates. These confounding factors introduced by the UI, however, would have impacted both conditions equally, because the control condition was simulated using the same infrastructure and development environment. Finally, for users in the control condition, Task 4 may have been deceptively tricky due to its impossibility. Nevertheless, the incorrect answers underscore a very real problem: Android users are not aware that they are unable to deny resources to applications that they are not using.

8.5 Conclusion

Android's existing permission models, ask-on-install (AOI) and ask-on-first-use (AOFU), are insufficient at fulfilling users' privacy desires and needs. Neither of the existing models account for contextual factors in their decisions to allow or deny access to sensitive data. Users want to protect their sensitive information, but have a hard time understanding when access to data is and is not being allowed. TurtleGuard adds both ease of use and functionality, including the ability to consider application visibility when specifying privacy preferences, which has been shown to be a strong contextual cue. In our study of TurtleGuard, we had participants perform permission-related tasks and compared their performance TurtleGuard with a control group using the default permission manager. Based on our results, we iterated on TurtleGuard's design, and then performed a validation experiment to confirm the validity of our changes. Our results show that users are significantly better at performing permission management tasks with TurtleGuard than the de- fault permission manager.

Acknowledgements

This research was supported by the United States Department of Homeland Security's Science and Technology Directorate under contract FA8750-16-C-0140, the Center for Long-Term Cybersecurity (CLTC) at UC Berkeley, the National Science Foundation under grants CNS-1318680 and CNS-1514457, Intel through the ISTC for Secure Computing, and the AFOSR under MURI award FA9550-12-1-0040. The content of this document does not necessarily reflect the position or the policy of the U.S. Government and no official endorsement should be inferred.

9. REFERENCES

[1] H. Almuhimedi, F. Schaub, N. Sadeh, I. Adjerid, A. Acquisti, J. Gluck, L. F. Cranor, and Y. Agarwal. Your location has been shared 5,398 times!: A field study on mobile app privacy nudging. In *Proc. of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 787–796. ACM, 2015.

[2] P. Andriotis and T. Tryfonas. Impact of user data privacy management controls on mobile device

investigations. In *IFIP International Conference on Digital Forensics*, pages 89–105. Springer, 2016.

[3] Apple. About privacy and location services for ios 8 and later. https://support.apple.com/en-us/HT203033. Accessed: March 4, 2017.

[4] A. Barth, A. Datta, J. C. Mitchell, and H. Nissenbaum. Privacy and contextual integrity: Framework and applications. In *Proc. of the 2006 IEEE Symposium on Security and Privacy*, SP '06, Washington, DC, USA, 2006. IEEE Computer Society.

[5] M. Benisch, P. G. Kelley, N. Sadeh, and L. F. Cranor. Capturing location-privacy preferences: Quantifying accuracy and user-burden tradeoffs. *Personal Ubiquitous Comput.*, 15(7):679–694, Oct. 2011.

[6] M. Bokhorst. Xprivacy. https://github.com/M66B/XPrivacy, 2015.

[7] CollegeDev. Donkeyguard. https://play.google.com/store/apps/details?id=eu.donkeyguard, 2014.

[8] S. Egelman, A. P. Felt, and D. Wagner. Choice architecture and smartphone privacy: There's a price for that. In *The 2012 Workshop on the Economics of Information Security (WEIS)*, 2012.

[9] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In *Proc. of the ACM Conf. on Comp. and Comm. Sec.*, CCS '11, pages 627–638, New York, NY, USA, 2011. ACM.

[10] A. P. Felt, S. Egelman, M. Finifter, D. Akhawe, and D. Wagner. How to ask for permission. In *Proc. of the 7th USENIX conference on Hot Topics in Security*, Berkeley, CA, USA, 2012. USENIX Association.

[11] A. P. Felt, S. Egelman, and D. Wagner. I've got 99 problems, but vibration ain't one: a survey of smartphone users' concerns. In *Proc. of the 2nd ACM workshop on Security and Privacy in Smartphones and Mobile devices*, SPSM '12, pages 33–44, New York, NY, USA, 2012. ACM.

[12] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: user attention, comprehension, and behavior. In *Proc. of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, New York, NY, USA, 2012. ACM.

[13] H. Fu, Y. Yang, N. Shingte, J. Lindqvist, and M. Gruteser. A field study of run-time location access disclosures on android smartphones. *Proc. USEC*, 14, 2014.

[14] N. Good. The Deadly Sins of Security User Interfaces. In M. Jakobsson, editor, *The Death of the Internet*, chapter 7.5, pages 398–415. John Wiley & Sons, 2012.

[15] Google. Normal and dangerous permissions. https://developer.android.com/guide/topics/permissions/requesting.html#normal-dangerous.

[16] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall. These aren't the droids you're looking for: retrofitting android to protect data from imperious applications. In *Proc. of the ACM Conf. on Comp. and Comm. Sec.*, CCS '11, pages 639–652, New York, NY, USA, 2011. ACM.

[17] L. Jedrzejczyk, B. A. Price, A. K. Bandara, and B. Nuseibeh. On the impact of real-time feedback on users' behaviour in mobile location-sharing applications. In *Proceedings of the Sixth Symposium on Usable Privacy and Security*, page 14. ACM, 2010.

[18] P. G. Kelley, S. Consolvo, L. F. Cranor, J. Jung, N. Sadeh, and D. Wetherall. A conundrum of permissions: Installing applications on an android smartphone. In *Proc. of the 16th Intl. Conf. on Financial Cryptography and Data Sec.*, FC'12, pages 68–79, Berlin, Heidelberg, 2012. Springer-Verlag.

[19] H. Lee and A. Kobsa. Privacy Preference Modeling and Prediction in a Simulated Campuswide IoT Environment. In *IEEE International Conference on Pervasive Computing and Communications*, 2017.

[20] B. Liu, M. S. Andersen, F. Schaub, H. Almuhimedi, S. A. Zhang, N. Sadeh, Y. Agarwal, and A. Acquisti. Follow my recommendations: A personalized assistant for mobile app permissions. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, 2016.

[21] B. Liu, J. Lin, and N. Sadeh. Reconciling mobile app privacy and usability on smartphones: Could user privacy profiles help? In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 201–212, New York, NY, USA, 2014. ACM.

[22] W. Mason and S. Suri. Conducting behavioral research on amazon's mechanical turk. *Behavior Research Methods*, 44(1):1–23, 2012.

[23] D. Mate. Permission master. https://play.google.com/store/apps/details?id=com.droidmate.permaster, 2014.

[24] M. McLaughlin. What is lineageos. https://www.lifewire.com/what-is-cyanogenmod-121679, 2017.

[25] H. Nissenbaum. Privacy as contextual integrity. *Washington Law Review*, 79:119, February 2004.

[26] K. Opsahl. Uber should restore user control to location privacy. https://www.eff.org/deeplinks/2016/12/uber-should-restore-user-control-location-privacy, 12 2016.

[27] G. Paolacci and J. Chandler. Inside the turk. *Current Directions in Psychological Science*, 23(3):184–188, 2014.

[28] S. Patil, Y. Le Gall, A. J. Lee, and A. Kapadia. My privacy policy: Exploring end-user specification of free-form location access rules. In *Proceedings of the 16th International Conference on Financial Cryptography and Data Security*, FC'12, pages 86–97, Berlin, Heidelberg, 2012. Springer-Verlag.

[29] S. Patil, R. Schlegel, A. Kapadia, and A. J. Lee. Reflection or action?: How feedback and control affect location sharing decisions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 101–110, New York, NY, USA, 2014. ACM.

[30] E. Peer, L. Brandimarte, S. Samat, and A. Acquisti. Beyond the turk: Alternative platforms for crowdsourcing behavioral research. *Journal of Experimental Social Psychology*, 70:153–163, May 2016.

[31] D. Reilly, D. Dearman, V. Ha, I. Smith, and K. Inkpen. "need to know": Examining information need in location discourse. In *Proceedings of the 4th International Conference on Pervasive Computing*, PERVASIVE'06, pages 33–49, Berlin, Heidelberg, 2006. Springer-Verlag.

[32] X. M. Repository. http://repo.xposed.info/, http://repo.xposed.info/.

[33] J. Ross, L. Irani, M. S. Silberman, A. Zaldivar, and B. Tomlinson. Who are the crowdworkers?: Shifting demographics in mechanical turk. In *CHI '10 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '10, pages 2863–2872, New York, NY, USA, 2010. ACM.

[34] J. L. B. L. N. Sadeh and J. I. Hong. Modeling users' mobile app privacy preferences: Restoring usability in a sea of permission settings. In *Symposium on Usable Privacy and Security (SOUPS)*, 2014.

[35] F. Shih, I. Liccardi, and D. Weitzner. Privacy tipping points in smartphones privacy preferences. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 807–816, New York, NY, USA, 2015. ACM.

[36] I. Shklovski, S. D. Mainwaring, H. H. Skúladóttir, and H. Borgthorsson. Leakiness and creepiness in app space: Perceptions of privacy and mobile app use. In *Proc. of the 32nd Ann. ACM Conf. on Human Factors in Computing Systems*, CHI '14, pages 2347–2356, New York, NY, USA, 2014. ACM.

[37] C. Thompson, M. Johnson, S. Egelman, D. Wagner, and J. King. When it's better to ask forgiveness than get permission: Designing usable audit mechanisms for mobile permissions. In *Proc. of the 2013 Symposium on Usable Privacy and Security (SOUPS)*, 2013.

[38] P. Wijesekera, A. Baokar, A. Hosseini, S. Egelman, D. Wagner, and K. Beznosov. Android permissions remystified: A field study on contextual integrity. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, pages 499–514, Berkeley, CA, USA, 2015. USENIX Association.

[39] P. Wijesekera, A. Baokar, L. Tsai, J. Reardon, S. Egelman, D. Wagner, and K. Beznosov. The feasibility of dynamically granted permissions: Aligning mobile privacy with user preferences. *arXiv preprint 1703.02090*, 2017.

APPENDIX

Permission	Explanation
call_phone process_outgoing_calls read_phone read_call_log add_voicemail write_call_log	Make and process calls as well as read information about call status, network information and previously made phone calls
camera	Access camera devices
get_accounts	Access to list of accounts
read_calendar write_calendar	Read and write events to the user's calendar
read_contacts write_contacts	Read and write to user's contacts
read_external_storage write_externalstorage	Read and write files to the user's external storage
record_audio	Record audio
access_coarse_location access_fine_location access_wifi_state	Read location information in various ways including network SSID-based location
read_sms send_sms receive_sms	Read SMS messages from the device (including drafts) as well as send and receive new ones SMS

Table 5: Sensitive permissions managed by TurtleGuard. Permissions grouped by a single explanation form the families used in our system to reduce the number of managed permission as discussed in Section 3.

Condition	Correct	Incorrect	All
Task 1			
control	2	3	2
experimental	2	4	2
Task 2			
control	2	3	3
experimental	2	3	2
Task 3			
control	2	4	3
experimental	2	3	2
Task 4			
control	4	2	3
experimental	2	2	2

Table 6: Median ease-of-use Likert scores for all tasks, conditions, and correctness in the validation experiment. Higher scores indicate more difficulty.

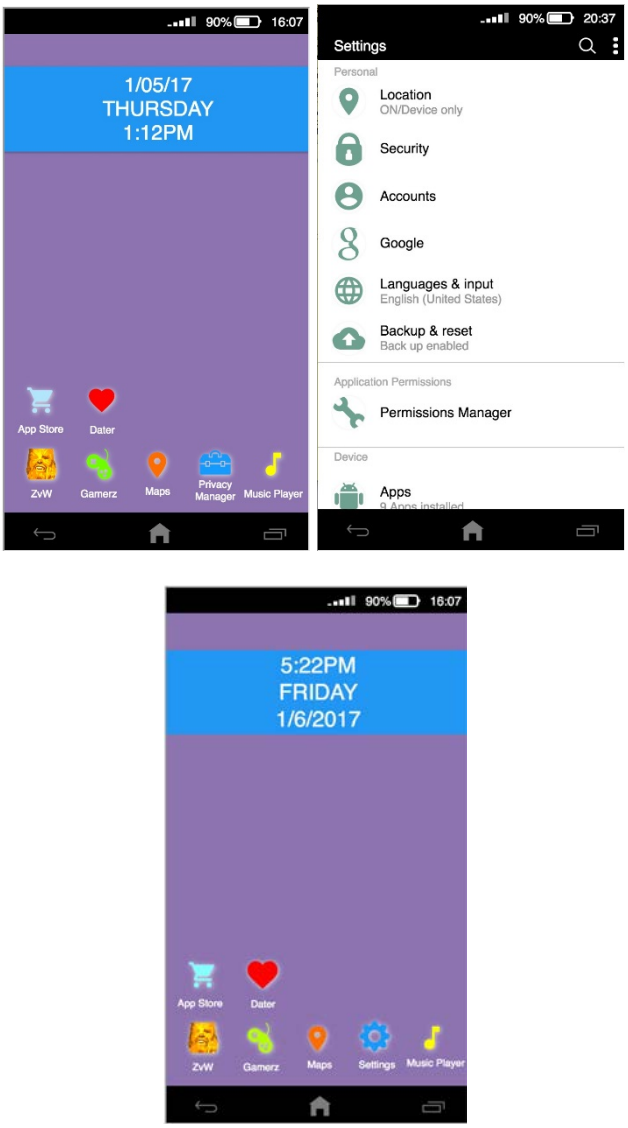


Figure 6: In the pilot experiment, TurtleGuard was launched via the icon labeled "Privacy Manager" (top left), but then added as a sub-panel to the Settings app, labeled "Permissions Manager," for the validation experiment (top right). In the *control* condition in the pilot experiment and both conditions in the validation experiment, the Settings app was accessible from the home screen (bottom).

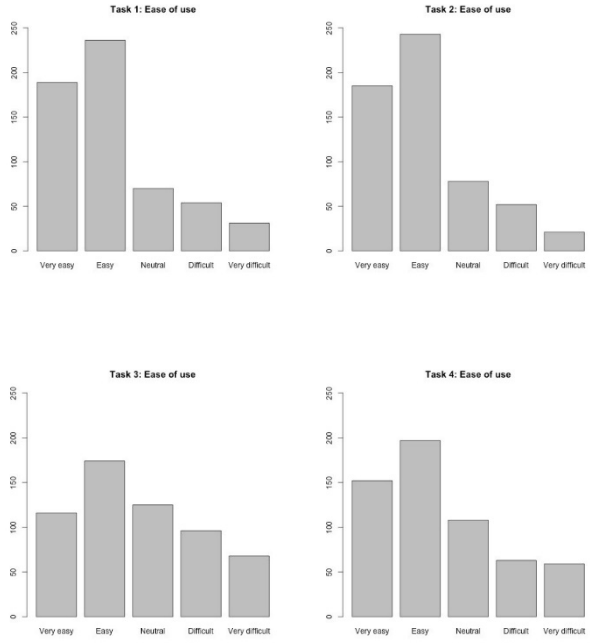


Figure 7: Ease of use histograms for each task (validation experiment)

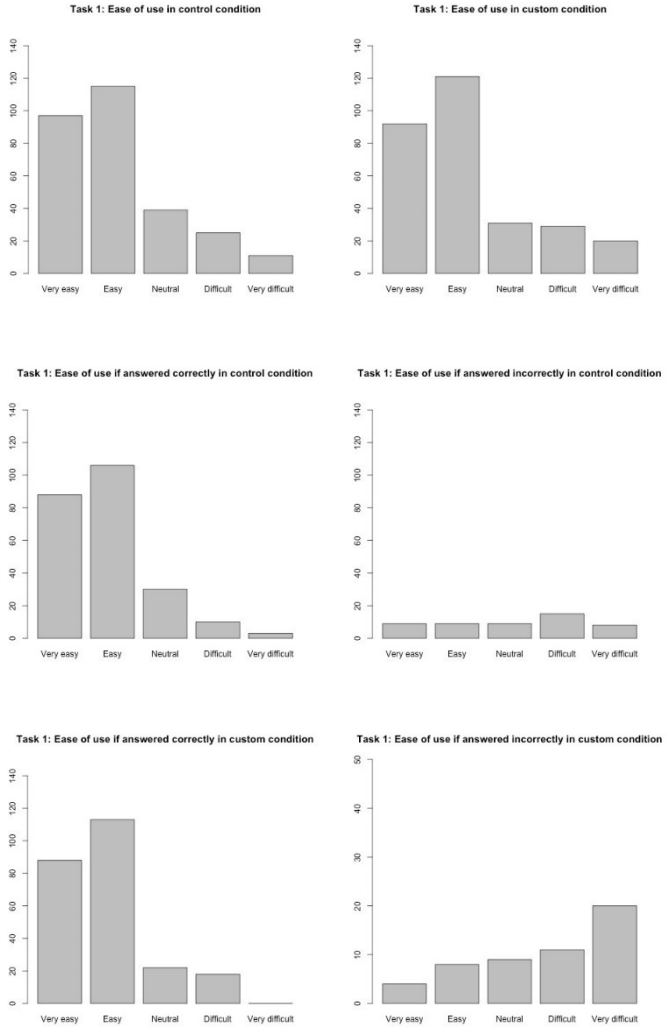
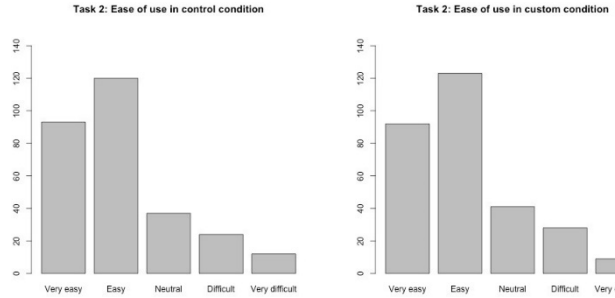
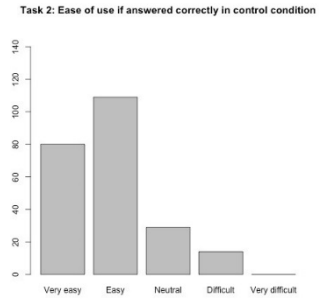
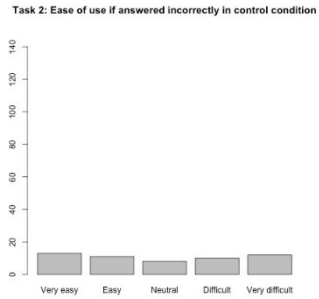
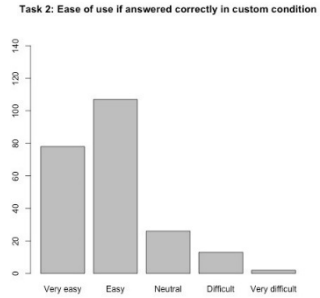


Figure 8: Ease of use histogram for Task 1 (validation experiment)









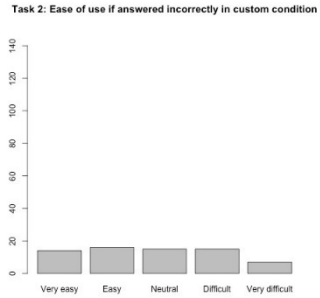
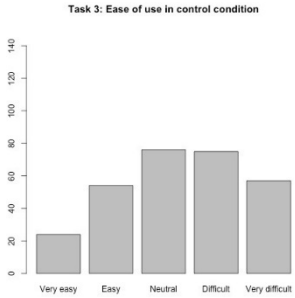
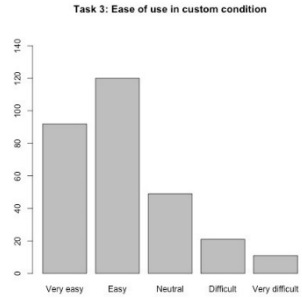
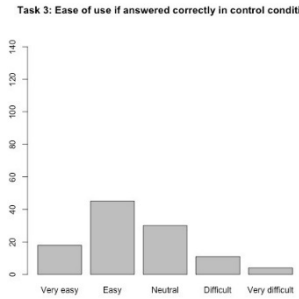
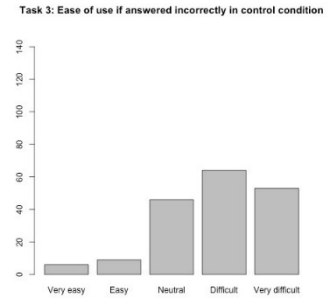


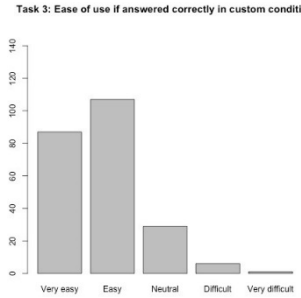
Figure 9: Ease of use histogram for Task 2 (validation experiment)











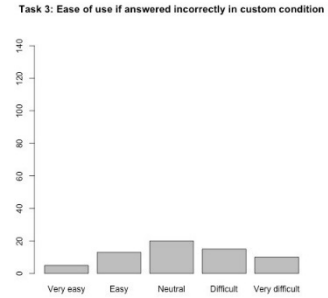


Figure 10: Ease of use histogram for Task 3 (validation experiment)

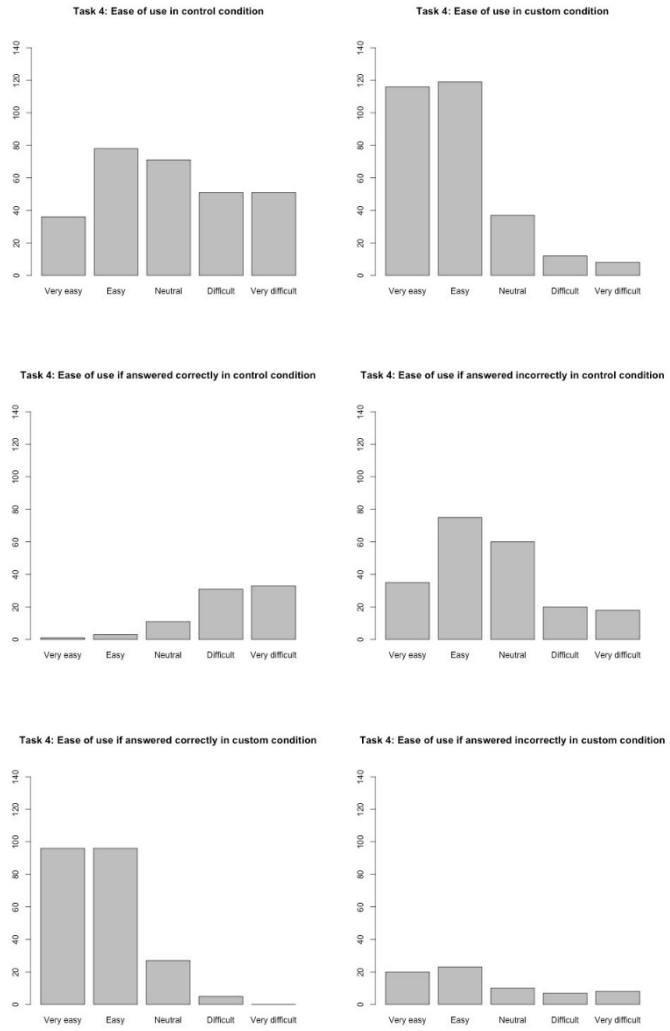


Figure 11: Ease of use histogram for Task 4 (validation experiment)

List of Symbols, Abbreviations and Acronyms

AOFU	Ask On First Use
API	Application Programming Interface
BYOD	Bring Your Own Device
CDMA	Code Division Multiple Access
ESM	Experience Sampling Method
GPS	Global Positioning System
GSM	Global System for Mobiles
HTTPS	Hyper Text Transfer Protocol Secure
ML	Machine-learning
NFC	Near field communication
OS	Operating System
SSID	Service Set Identifier
SMS	Short Message Service
SVM	Support Vector Machines
UI	User Interface

Glossary of Terminology

BYOD Bring Your Own Device Practice where employees and workers can bring their own mobile devices into the workplace to be used for both work and personal purposes.

Background Application Application that is running on a mobile device but is not readily visible to the end user.

Classifier The Classifier is a system that looks at user preferences (privacy preferences in the case of this report) and then recommends actions to perform on the behalf of the end user (e.g. deny access to a specific application for contact information). Over time, the classifier learns how to correctly *classify* user preferences and associate these with actions that it should perform for the end user.

Android The operating system at the time of this writing that is developed by Google used on mobile phones.

Dashboard The user interface we provided for allowing users to have control over the privacy settings for their applications on the Android phones

Lo-Fidelity prototypes Lo-Fidelity prototypes are prototypes created for user testing that display minimal information and are not entirely implemented.

Hi-Fidelity prototypes Hi-Fidelity prototypes are prototypes created for user testing that look and in some cases act as if they are implemented, however they still lack certain functionality that would make them actual implementations.